8-2021

System Integration of a Tour Guide Robot

Suhasa Prabhu Kandikere

System Integration of a Tour Guide Robot

by

Suhasa Prabhu Kandikere

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical Engineering

Supervised by

Prof. Ferat Sahin
Department of Electrical and Microelectronic Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology. Rochester, New York

August 2021

Approved by:

Prof. Ferat Sahin, Professor
Thesis Advisor, Department of Electrical and Microelectronic Engineering

---

Prof. Jamison Heard, Assistant Professor
Committee Member, Department of Electrical and Microelectronic Engineering

---

Prof. Gill Tsouri, Professor
Committee Member, Department of Electrical and Microelectronic Engineering

---

Prof. Ferat Sahin, Professor,
Department Head, Department of Electrical and Microelectronic Engineering

Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

System Integration of a Tour Guide Robot

I, Suhasa Prabhu Kandikere, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Suhasa Prabhu Kandikere

Date

Dedication

I dedicate this work to my loving family who have supported me all the time...

Acknowledgments

I am extremely grateful to Prof. Ferat Sahin for his invaluable support and guidance without which carrying out this experiment would not have been possible.

I would like to extend my gratitude to Celal Savur for his valuable help and suggestions in analyzing and debugging problems in this experiment.

I would like to acknowledge the help of Anmol Modur for setting up the robot with sensors and designing its CAD Model.

I would like to thank Shravan Krishna for his valuable insights in the navigation aspect of this experiment.

I would like to thank the members of the Multi-agent Bio Robotics Laboratory for providing me with insights on Robot Operating System and SLAM.

I would also like to extend my gratitude to the Rochester Institute of technology for providing me with the resources and tools necessary for carrying out this experiment.

I am deeply indebted to my parents for their unwavering support in pursuing my Master's.

Abstract

System Integration of a Tour Guide Robot

Suhasa Prabhu Kandikere

Supervising Professor: Prof. Ferat Sahin

In today's world, people visit many attractive places. On such an occasion, It is of utmost importance to be accompanied by a tour guide, who is known to explain about the cultural and historical importance of places. Due to the advancements in technology, smartphones today have the capability to help a person navigate to any place in the world and can itself act as a tour guide by explaining a significance of a place. However, the person while looking into his phone might not watch his/her step and might collide with other moving person or objects. With a phone tour guide, the person is alone and is missing a sense of contact with other travelers. therefore a human guide is necessary to provide tours for a group of visitors. However, Human tour guides might face tiredness, distraction, and the effects of repetitive tasks while providing tour service to visitors. Robots eliminate these problems and can provide tour consistently until it drains its battery. This experiment introduces a tour-guide robot that can be used on such an occasion. Tour guide robots can navigate autonomously in a known map of a given place and at the same time interact with people. The environment is equipped with artificial landmarks. Each landmark provides information about that specific region. An Animated avatar is simulated on the screen. IBM Watson provides voice recognition and text-to-speech services for human-robot interaction.

List of Contributions

- The base of the Wheel Chair is interfaced with different sensors. the wheelchair is made to navigate autonomously in the given environment (3rd Floor KGCOE) that contains Artificial Landmarks such as ArUco markers or April tags.
- Systems such as Robot Operating System, Unity, and IBM Watson which work independently of each other are integrated using WebSockets to function as a single system.
- An autonomous wheelchair is integrated with these systems to function as a tour-guide robot.
- A dataset of the 3rd floor Kate Gleason College of Engineering, RIT is provided. The dataset includes 2D laser scan, fused odometry, encoder, and imu values. This data can be later used for Simultaneous localization and mapping (SLAM) or testing other localization algorithms.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

tourists visit many different places all over the world. They may visit one of the wonders of the world or they might visit a monument that holds historical significance. On such an occasion they are accompanied by a tour guide. The tour guide leads tourists to many different places and explains the importance of a place. Due to the advancements in technology in recent years, a smartphone can be used as a guide for a person. The phone uses GPS for navigation and can lead a person to a destination. Assistants such as Apple's Siri, Google's Assistant, and Amazon's Alexa that are available on the smartphone can explain the importance of a place to a person. However, a person while looking into the phone can crash into other moving people or might trip and fall if he/she is not careful, the person might feel lonely with the phone guide, elderly people might have difficulties operating a smartphone, and disabled people in a wheelchair might not be able to maneuver in the path shown on the phone such as a staircase, etc. As a result, a human tour guide can provide tours for a group of visitors leading them carefully to places of interest and explain about the importance of the place. However, a tour guide may feel tired, distracted, and get bored due to repetitive tasks. During these times, a robot can provide a tour to a group of visitors. Such a robot is called a tour-guide robot.

A robot is a programmable machine, which can help in reducing the workload of humans in different tasks. there are various kinds of robots such as Mobile robots, Arm robots, legged robots, and so on. Each has its own function and advantages. This experiment focuses solely on a mobile robot. A mobile robot uses its sensors to perceive the environment and move around the environment using its legs, wheels, tracks. This experiment focuses on a mobile robot with wheels. This robot is used to implement the functionality of a tour-guide robot.

1.1 Navigation

Many types of research are going on in the field of robotics and artificial intelligence [6, 11], which has led to the development of many different algorithms which is required to make a robot autonomous. A tour guide robot requires a fundamental navigation technique to avoid obstacles in the environment and maneuver safely to its destination and an interactive Speech system for Human-Robot interaction. navigation requires sensors to perceive the environment and a speech system requires access to a microphone and speaker.

1.2 Speech System

Thanks to the inventions and discoveries in today's world, Natural language processing, and Text-to-Speech system is well developed to be used for human-robot interaction. Natural language processing is a branch of Artificial Intelligence, which deals with human speech synthesis which is machine-understandable. Speech recognition converts each word spoken by the human into a machine-understandable format. As the name suggests Text-to-speech

is a module that can convert text to speech. One such example of natural language processing is IBM Watson. The guide robot needs to be equipped with a microphone and a speaker to interact with people. The human speech is read through the microphone and a reply is given through the speaker.

1.3 Components

This section discusses the hardware part of the robot that is required in the autonomous navigation of the robot. The robot in this experiment has two 12v Batteries powering the two wheels which are about 13" in diameter, it also comes with two castor wheels. The sensors used are as follows:

1.3.1 Lidar

Figure 1.1: YDlidar G4 [42]

Lidar stands for Light Detection and Ranging. It uses light in the form of pulses to determine the distance from the obstacle to the sensor. There are 2D lidar and 3D lidar available, The Lidar used in this experiment is a YDlidar g4. It is a 2D Lidar that has a 360° sensor bearing and has a maximum range of 16 meters.

1.3.2 Camera

Figure 1.2: Intel Realsense D435i [43]

The robot in this experiment is equipped with an Intel RealSense D435i. This module has a maximum resolution of a 1080p monocular camera. It also has an Inertial Measurement Unit, which provides linear acceleration and angular velocity on three axes.

1.3.3 Wheel Encoder

The robot comes equipped with a CUI absolute encoder that has a resolution of 4096 pulses per revolution. The gear ratio of motors on the robot is 29:1. Therefore, the total number of ticks per revolution of the wheel is approximately 118700.

1.4 Software

ROS provides a powerful set of tools that are used in this experiment for making the robot navigate autonomously. Robot Operating System is a widely used tool in the field of robotics that allows developers to set up the robots and It also exposes its various API to communicate with different third-party software.

Unity game engine provides an Avatar interface for speech systems like IBM Watson and also helps in communication with ROS.

Chapter 2

Literature review

In recent years, many kinds of research have been conducted on different aspects of robotics and artificial intelligence, which has led to the development of many tour guide robots with each having its own advantages.

2.1 Tour Guide Robots

The purpose of a Tour guide robot is to give tours to people in different places like museums. Places like Museums are often filled with exhibits that are expensive and are crowded with people. The two fundamental blocks of a tour guide robot are, the robot must be able to autonomously navigate the environment safely and reliably, it must be able to interact with people around it. One such robot was presented by Sebastian Thrun et al [10] called RHINO. this robot was able to travel at speeds compared to the walking speed of the people around it. This robot had a web interface that allowed people around the world to virtually move around the museum by giving goal points to the robot. When the tour is finished, the robot moves to the entrance of the museum awaiting its next set of visitors

Minerva is a second-generation tour-guide robot that was a revision to Rhino, which was a tour guide robot introduced in mid-1997. A year later Minerva was introduced to people with a lot of features that Rhino lacked. Minerva is able to create maps of different places.

Figure 2.1: Rhino: A tour guide robot [35]

Since it is an indoor tour guide robot it uses the mosaics on the ceiling to localize itself. It uses a high-level control for path planning, where the robot maneuvers in feature-rich spaces. It also has a system similar to finite state machines, where the robot can handle exceptions that occur during the tour. If it senses a voltage drain in the battery it returns to its charging station. There is also a good chance that the localization of Minerva is inaccurate, at these times, the robot tour is temporarily suspended. The robot also comes equipped with an interactive interface which is an overhaul from Rhino for human-robot interaction. It also comes equipped with a web interface that portrays the map and the location of the robot for the virtual tours. Sebastian Thrun et al [8] introduce a tour-guide

Figure 2.2: Minerva: 2^{nd} generation tour guide robot [36]

robot that can navigate at the same speed as Rhino, and maneuver safely in a crowded indoor environment specifically a museum.

Skycall is a tour guide quadrotor designed and developed by MIT [9]. It helps college freshman find their way to their classes. Skycall app can be downloaded to any smartphone. This app helps the users to communicate with the quadrotor. Opening the app lets the quadrotor navigate to your position and by sending it the number on the classroom, the quadrotor navigates inside the building and can lead the freshman to their destinations.

A tour guide Humanoid robot by the name ASIMO was introduced by Honda in the year 2000 [10]. this robot has the capability to interact with people but is limited to 100

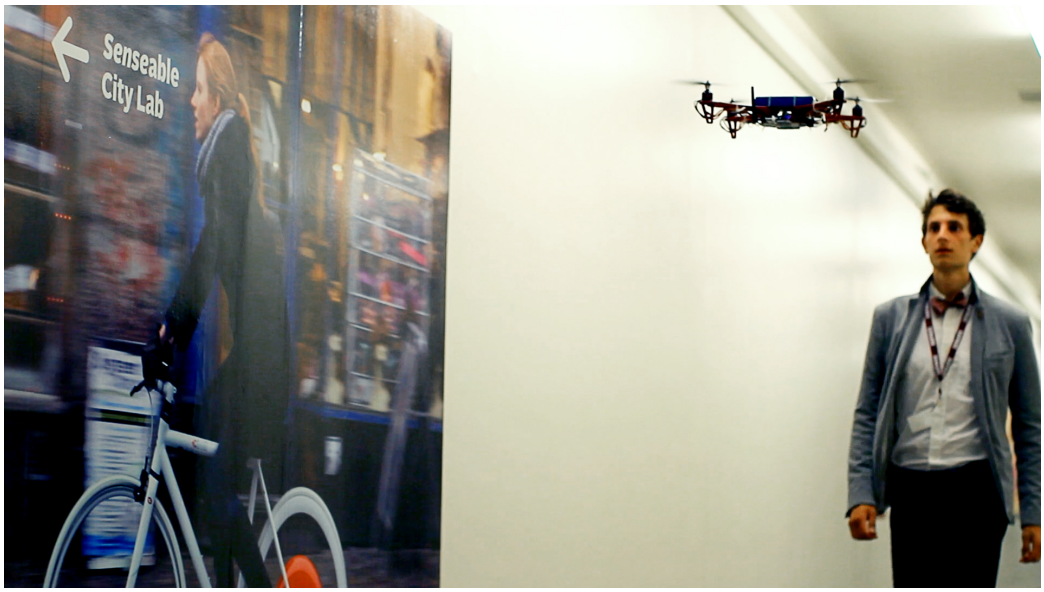

Figure 2.3: Skycall [9]

questions. The questions were listed on the touch panel of ASIMO. This robot could not differentiate between people raising their hands to ask questions and people who wanted to capture photos. It would freeze mid-way of answering questions if it noticed any other person raising their arms.

When people explore new places, it is very likely that they visit tourist places in that area. Exploring the site without any guide will be exhausting to people. To solve this problem, there are many tour guide robots, one such tour guide robot is called Eddie. Eddie has Kinect as its main sensor, typically it's used in XBOX, but due to the depth point clouds it provides, it has been used in many different fields. It's used a lot in the robotics domain reason being, it consists of an RGB camera, depth-sensing system, and microphone. Kinect can detect a person by tracking the human skeleton available in Kinect. The main advantage of Kinect is that it can differentiate between objects that have the same color due to its depth sensor. Eddie can detect RFID (Radio-frequency identification) tags that are attached to the

Figure 2.4: Asimo [40]

objects. These tags are identified using electromagnetic fields. Using these Tags, Eddie can detect objects in the museum and play the audio file stored for that respective tag. A person can observe the current location of Eddie through a website. These tags also help in the localization of the robot. The robot follows the user if he/she gives a command "follow me" and stops if he/she commands the robot to stop. This operation requires that Eddie must be a certain distance away from the robot. These tags will help in updating the robot's location on the website. This was introduced by Wazzan et al [22].

The experiment introduced by Chung et al. [23] is about navigation, localization, path planning, and autonomous control of a tour guide robot in detail. The kinematics model

of the tour guide robot introduced calculates the average angular velocity of the wheels of the robot, which can then be used to find the linear velocity. This robot has a facial expression system. The robot uses RFID tags to localize itself, the four tags are placed in the environment, RSSIs data are received from these tags, which is used to calculate distances from these tags to the robot, and by using least square methods the initial robot location is calculated. Additionally, the robot is equipped with a laser scanner, by fusing RFID and laser scan data using an extended Kalman filter, the robot can be localized with good accuracy. Assuming the robot moves in a straight line, obtaining observations of the robot's position for certain time instances, the robot's orientation is calculated by using the initial position obtained by the RFID tags and the new position obtained by the straight-line equation. Dynamic programming is used to solve the shortest distance path for the robot, this dynamic programming starts from a terminal state and traces back the route to the robot's location. Using a High-level autonomous robot control Petri-net, the robot navigates its environment avoiding obstacles by planning the route safely to its destination. The robot is equipped with two-color cameras mounted on the facial expression system to calculate the position of the approaching visitors and interact with them with the standard Chinese language. Experimental results and simulations prove that the method proposed for navigating a tour guide robot is effective and successful in accomplishing its tasks.

2.2 SLAM

One fundamental rule for a tour guide robot is it has to navigate the environment autonomously. to achieve this, the robot needs a map of the environment. The robot has to map the environment and also has to be aware of its position. This problem in robotics

is known as Simultaneous Localization and Mapping. Below are few related works on solving this problem.

Robot localization is a required part of SLAM, many types of research have been conducted in this area, to efficiently localize a robot. A new algorithm that fuses the laser slam and visual slam, to obtain a highly accurate robot position in an environment is introduced by Chan et al. [20]. In this experiment, a map of the environment is extracted by Laser slam and visual slam. The SLAM algorithms are fused after extracting the robot trajectory from the maps. After extraction, the coordinates in images are scaled in such a way that these two trajectory images have the same height and width. After which the aligned angles of the images are calculated. This is done by extracting the coordinates of the trajectories at the same timestamp from both slam techniques and calculating the distance between them. The loss here is the distance. In order to minimize it, images are rotated around the z-axis until minimum loss. Since this is a continuous trajectory, the trajectories may not consist of sharp turns, but if it exists, it needs to be filtered which is carried out with a curvature filter. The filter extracts the previous, present, and future coordinates of the trajectory at the present time step and calculates the distance between all the points and the angle at the present time step. If the angle is small, the future point is filtered. Now after smoothing out the trajectory the two trajectory images are passed through a pyramid filter. This filter scales the images by a scaling ratio '1/r' and rounds off the points, to upsample the images to their original dimensions, the images are scaled by 'r'. Performing this operation filters out the noise in the images. After the two images are processed, the essential matrix is calculated from both the images along with the transformation matrix used in the mapping.

Mapping and localization of a robot is a well-known problem in the field of robotics. The mapping is a process in which the robot tries to create a blueprint of the unknown environment with the help of its sensors. There are many popular algorithms in robotics, which are used to solve slam problems. Saman et al. [16]. go on about implementing an extended Kalman filter to solve the slam problem. Kalman filters are used to predict the position of the landmarks with odometry sensors and update using the values from the sensors using mathematical equations. The most important part of these equations is Kalman gain, which gives information about the quality of sensor measurements. However, the Kalman filter is only valid to linear equations, In the real world all the measurements are non-linear, therefore extended Kalman filters are used in the prediction and update steps of the Kalman filter. software and hardware implementation is introduced. In a software implementation, the extended Kalman filter is implemented using the required equations in python and is tested on the library data set available on the web. The hardware implementation involves placing landmarks in the given environment and allowing the robot to maneuver in the environment to test out the accuracy of the algorithm. Since odometry in robots was different from software, there exist many errors, which are then fed into EKF (extended Kalman filters) along with laser scan data to create a well-structured map and localize the robot as introduced by Saman et al. [16]. This experiment yielded good results with fewer inaccurate measurements. EKF slam resulted in a good map even though the measurements had a lot of noise.

A system for occupancy grid mapping using a mobile robot equipped with an ultrasonic range finder is discussed by Hadji et al [4]. In this experiment, an ultrasonic ping sensor is

mounted on the robot and has the ability to sweep 90 degrees on either side. The inverse sensor model is used to create a grid map from the data available from the ping sensor at the position provided by odometry sources. the cells in the map occupied by obstacles have a probability of 1 and cells with no obstacles have a probability of 0.

2.3 Path Planners

Once the map is constructed, the next step is for the robot to plan the path from source to its destination. There are two types of planners. They are global planners and local planners. Global planners plan a route from source to destination and local planners provide control commands to robot such as velocity, acceleration to follow the path constructed by global planner and avoid obstacles at the same time.

Naotunna et al. [5] compares different local planners available in ROS for differential drive heavy robots based on their goal reaching accuracy, the path chosen to reach the destination, and time consumption. The comparison of DWA planner, TEB local planner, and EBand local planner is shown. The experiment is carried out with obstacles and without obstacles. Eband local planner needs more inflation around the obstacles as it makes the robot move very close to the walls. However, the paper concluded that DWA and TEB local planners both have an effective path planning capability. DWA planner is smooth and consumes less time to reach the goal, however, TEB local planner performs significantly better than the other two planners when there are obstacles in this path.

Fox et al. [6] describes a planner for avoiding obstacles and maneuvering to reach the goal. Dynamic Window Approach planner is introduced for this purpose. This planner samples from the velocity space of the robot (v, w). For each sampled value from this

space, a simulation is performed to understand the effect of applying these values to the robot. trajectories produced by these sampled values are then awarded a score depending on the distance from the obstacles, distance from the goal, distance to the global path. the trajectory with the highest score is selected and the values (v, w) responsible for that trajectory is provided as input to the robot as described by Fox et al [6].

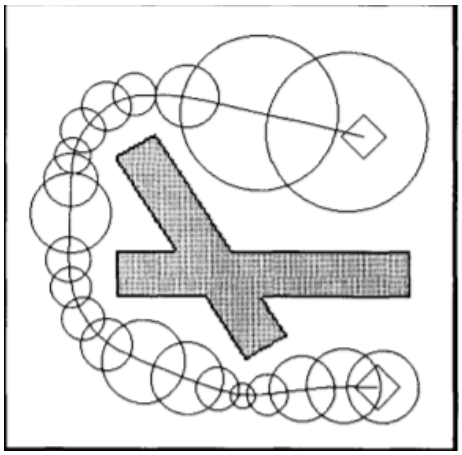

Figure 2.5: Bubbles along the path [11]

Global planners and local planners are responsible for navigating the robot safely and reliably in the given map. Quinlan et al introduce an Eband planner that relaxes the global planner[11]. As the name suggests, EBand stands for Elastic Bands, which takes into account the path planning and control capability of the robot. The global planner plans a path from a start to the destination point. The Eband planner deforms this path in real-time when it comes near an obstacle. The main concept here is the bubble around some point on the robot. The bubble around the robot informs about the distance of the obstacle from that robot. The deformed path has to be inside the bubbles created by this planner for a collision-free path. To improve the path shape, contraction force and repulsion force are implemented. the contraction force is the tension in the stretched elastic band. The

repulsion force is from the obstacles to avoid a collision.

2.4 Fiducial Markers

The robot must know when it reaches a region of interest to explain about that place. Fiducial markers help the robot in identifying those places in the environment.

A new visual fiducial system is described that improves upon the previous systems by incorporating the fast line detection system and avoids occlusion, lens distortion to its maximum extent. A graph-based image segmentation algorithm is proposed which allows precise estimation of lines in the image. A digital coding system is introduced that is not susceptible to false positives. The detection method introduced by this system has a better localization accuracy as proposed by Olson et al [2].

An experiment on ArUco markers, to localize the robot globally and locally is introduced by Babinec et al. [21]. The experiment is conducted with a webcam and HDR camera. The webcam provides a high resolution and HDR provides a high dynamic range. ArUco markers are 7x7 square markers, in which the outer rows and columns are black, each marker is represented by 5 words, each represented by 5 bits. It uses 3 bits for encryption and the rest 2 bits carry information about the markers. Since there are 4 combinations for 2-bits, and there are 25 bits in total for each marker, there is a total of 1024 markers available. Since it has a limited number of markers, it can be used for localizing a robot in a small area. ArUco package detects the markers and their respective identity number, this package provides the position relative to the optical center of the camera and a quaternion consisting of rotation information, which then must be converted into Euler angles to determine the robot's location and orientation. The markers are available in different sizes, the

experiment was conducted using 10 and 16cm markers. Errors were minimized with the addition of more landmarks, the webcam did not detect markers when there was a strong backlight and the HDR camera had difficulties when detecting smaller landmarks, since it has a wide-angle lens.

2.5 Speech system

Once the robot reaches a place, it has to explain about the importance of a place. Natural Language processing plays a significant role in this experiment by interacting with visitors.

The speech recognition system is an important part of a guide robot. The speech from a user is an analog signal detected by the computer. Each person has his/her unique voice and to differentiate this is not a trivial task. The overview of a speech recognition system is introduced by Rawat et al. [25]. There are two types of speech recognition systems, speaker-dependent, and speaker-independent speech recognition systems. The Speaker-dependent version must be trained with a large library of voice datasets. However, the speaker-independent version does not dependent on the voice of a person, hence these have limited vocabulary. After the conversion to digital signals, features must be extracted from these speech signals, which removes redundant information and other noise-related components which makes the signals contain only essential information that is required to recognize speech. The most popular features for speech signals are the Mel frequency cepstral coefficient (MFCC), which is commonly used to recognize numbers spoken through a telephone. There is also Perceptual linear prediction defining the human auditory system in an effective manner and Linear predictive code which assumes that the sound is produced at the end of the tube. Decoding of these speech signals requires the phonemes and the words

in the dictionary to be compared to the processed analog signals using the Hidden Markov model. This recognition of speech can be fed through text-to-speech recognition resulting in many useful applications such as controlling automated appliances in the home, etc.

AlShu'eili et al. designed a home automation system that works on voice commands for elderly and disabled persons [44]. The voice is captured at an 8 kHz rate and converted into digital code, then it is processed and converted to analog signals in a microcontroller, which then is analyzed in a computer, and then the commands are sent to the home appliance devices wirelessly using a ZigBee RF module.

2.6 ROS and Unity Integration

this section deals with integrating ROS and Unity game engine using ROSbridge library through WebSockets.

When it comes to working with virtual reality, the Unity engine provides robust support for the integration of headsets and games. ROS on the other hand provides numerous tools to developers to work on robots. A system is developed that integrates these two environments using WebSockets. Websockets allow the software to be running on different hardware but require the systems to be connected to the same reliable network for data exchange. The system is tested by teleoperating the robot through a virtual reality-based interface written in Unity as introduced by Codd-Downey et al [3].

Hussein et al. [7] integrates ROS and unity for vehicle simulation using the ROS bridge library available in ROS. the comparison of the performance in simulation and in real life for turtle bot 3 is shown. Additionally, the results were also compared to the output obtained on the gazebo [7]. Both the libraries use a JSON API to encode and decode data that are

passed through WebSockets.

In this experiment, the robot has multiple subsystems each performing its own tasks such as autonomous navigation, speech synthesis and an interactive avatar for human-robot interaction. This experiment chooses specific modules independent of each other and integrates them to provide a functionality of a tour guide. these modules include Robot Operating System for autonomous navigation, Unity for avatar representation, and IBM Watson for speech synthesis. The advantage of such a system is that each system can be run on a different machine for increased performance.

Chapter 3

Proposed Method

This experiment describes the work that went into implementing an autonomous tour-guide robot. The entire system was simulated and tested before actual implementation on the real robot.

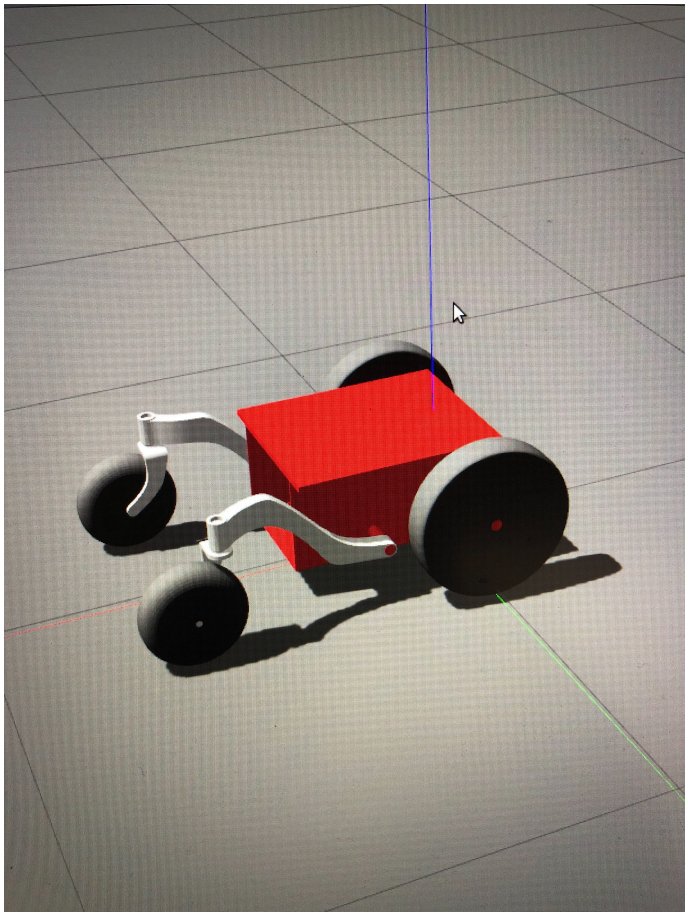

Figure 3.1: Wheel Chair simulated in Gazebo

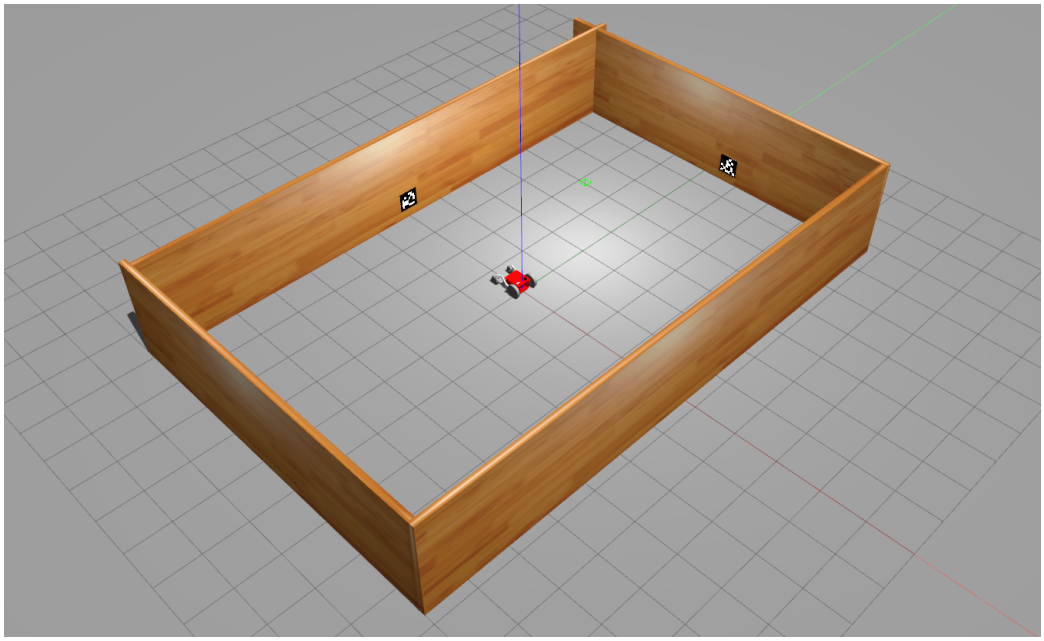

Figure 3.2: Simple test environment in Gazebo

3.1 Simulation

ROS framework provides numerous tools to work on a simulated robot if a user doesn't have access to a real robot. The simulation environment can be chosen based on the type of experiment that is being conducted. If a simulation requires people walking around, the Coppelia sim can be used. If the simulated environment needs trees, high-quality textures, a game engine such as Unity or Unreal engine can be used. However, ROS is available with a default simulation environment called Gazebo. This experiment mainly focuses on the Gazebo engine. This engine is built on OpenGL which is a specification and allows the program developed to be cross-platform. The simulation engine has all the physics implemented and can be altered as well. To be able to simulate the Permobil c300, It was first designed in CAD software and later converted into a Unified Robot Description Format (URDF in short). Gazebo understands URDF and allows the model to be simulated in the

environment. Gazebo supports creating and tweaking the environment easily. Due to this, it is the most widely used simulation environment. ROS provides a framework to write controls to any robot. There are many prebuilt controls that can be used as a plugin. One such plugin called the Differential Drive plugin is used.

After setting up the simulation environment, the rest of the process is the same as the one conducted on the real robot. The simulation is depicted in the figures 3.1 and 3.2

3.2 Overview of System Block Diagram

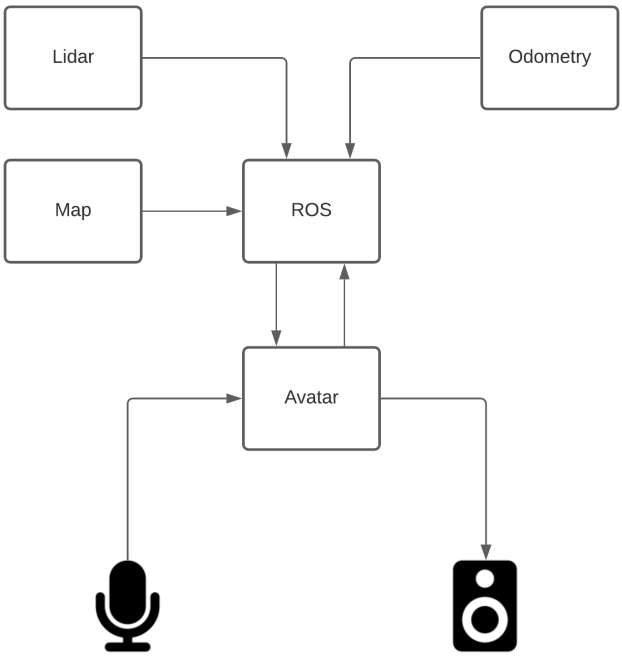

Figure 3.3: Overview of system

The figure 3.3 shows a simple block diagram of a tour guide robot. ROS short for Robot operating system takes map, laser scan and odometry as input to autonomously navigate the robot in an environment. An animated avatar is displayed on the screen for human-robot

interaction. The next sections are arranged as follow:

- Hardware: this sections introduces the lower level hardware on the robot such as motors, encoders.
- Controls: implementation of a controller for driving the robot straight
- Software: software that provides the infrastructure for navigating the robot autonomously and the game engine used for avatar representation.
- Navigation: explains about the algorithms necessary to make the robot autonomously navigate the environment
- System Integration: deals with integration of different systems to make a functional tour guide robot.

3.3 Hardware

The base of the Tour guide robot is salvaged from Permobil C300 WheelChair. It consists of two 12v DC batteries, sufficient to power the entire hardware on the robot. The robot consists of a lidar, Intel Realsense camera D435I with an IMU and wheel odometry. The lidar is mounted in front of the robot so that it can detect obstacles with lower heights. This limits the lidar to sweep the area in front of the robot instead of the 360 degrees around it.

3.3.1 setup

The initial hardware setup of the robot consisted of a raspberry pi 3, teensy 4, Sabertooth motor driver, YDLidar, Intel Realsense D435i depth camera, router, and a CUI encoder.

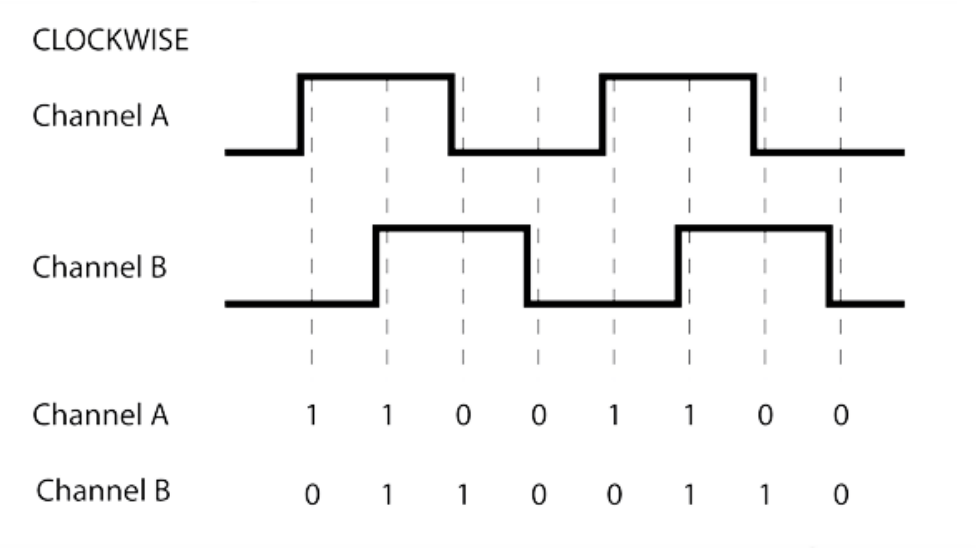

Figure 3.4: signals from encoder motor turning in forward direction [37]

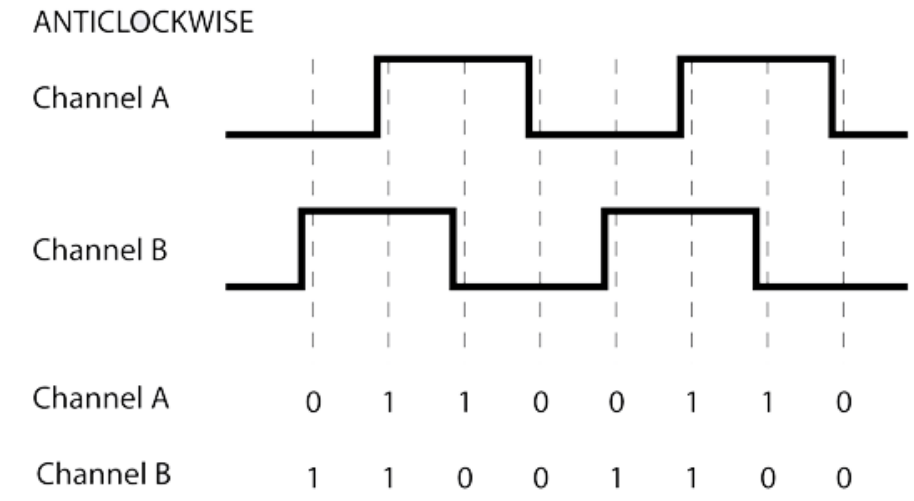

Figure 3.5: signals from encoder motor turning in reverse direction [37]

3.3.2 Encoders

The Encoder used in this experiment is a quadrature rotary encoder. rotary encoders are devices used to measure the degree of rotation. A quadrature encoder is used in applications that require bidirectional sensing capabilities. In this experiment, the wheels of the

robot turn both forward and backward. Hence, a quadrature encoder is used. A quadrature encoder outputs two square signals, they are channel A and channel B signals as shown in the above figure. these two signals are out of phase with each other. The direction of rotation of the wheel can be known by observing the states of the signal at any given instant of time. In this experiment, The robot is equipped with a CUI quadrature encoder. Counting both the rising and falling edge of both channels per rotation of the motor shaft, CUI encoders produce 4096 pulses. The motors are connected to the wheel through gears, which increases the torque required to rotate the wheel. The gear ratio of the gearbox is 29:1. This means for every rotation of the wheel, the motors rotate 29 times. Therefore, the total number of encoder ticks per revolution of the wheel is given by the formula:

$$\begin{aligned} Ticks &= Pulses \times 29 \\ &= 4096 \times 29 \\ &= 118784 \end{aligned} \tag{3.1}$$

In the above equation *"Ticks"* correspond to the number of ticks per revolution of the wheel. *"Pulses"* refer to the rising and falling edges of the signal in both channels and 29 is the gear ratio.

After calculating the ticks per revolution of the wheel, the next step in the process is to calculate the position and velocity of the robot. They are calculated as shown below:

$$velocity(RPM) = \frac{60 \times differential\ ticks}{118784 \times differential\ time} \tag{3.2}$$

In the above equation *"differential ticks"* correspond to ticks from the encoder in a specified amount of time and *"differential time"* corresponds to the time difference between the previous ticks and present ticks. In this experiment differential time is set to around 20milliseconds. *"velocity"* is the velocity of the wheel in RPM. The linear speed of the wheel is calculated from the rotational speed of the wheel as follows:

$$V = \frac{\omega \times r \times 2\pi}{60} \tag{3.3}$$

In the above equation 'v' denotes linear velocity of the wheel in M/s, 'ω' denotes angular velocity of the wheel in RPM and 'r' denotes the radius of the wheel in M.

Position and orientation calculation are as follows:

$$X = \frac{V_L + V_R}{2} \times dT \times cos(\theta) \tag{3.4}$$

$$Y = \frac{V_L + V_R}{2} \times dT \times sin(\theta) \tag{3.5}$$

$$\theta = \frac{V_R - V_L}{D} \times dT \tag{3.6}$$

In the above equation *"X"* corresponds to the position on X-axis, *"Y"* corresponds to the position in the Y-axis, "θ" corresponds to the orientation with respect to Z-axis, "V_R" corresponds to the velocity of the right wheel in M/s, "V_L" corresponds to the velocity of the left wheel in M/s and *"D"* corresponds to the distance between two wheels on the robot.

The position and velocity of the robot are then fed to the Kalman filter block for data fusion to obtain accurate odometry.

The heart of the robot is Raspberry pi, running the robot operating system on ubuntu

melodic. Teensy is interfaced with the wheel encoders. The data from teensy is sent serially to the raspberry pi using UART communication. The YDLidar is interfaced with raspberry pi. The robot hosts its network through a router. any computer that can connect to its network can access all the peripherals of the robot. Teensy is also interfaced with a sabertooth motor driver, capable of driving two 12v DC motors.

3.4 Controls

The robot is Non-Holonomic, meaning it can move forward on the x-axis and cannot move in the y or z-direction. The robot has to align itself to the target and then start to move towards the target. The robot's position and orientation are calculated using the ticks provided by the encoder of the motors.

As mentioned in the previous section, The robot comes equipped with two 12v Dc motors with CUI encoders. The encoders provide 118,784 ticks per revolution of the wheel. The wheel's diameter is around 13 inches. The robot when moving forward doesn't move in a straight line. This is because of many reasons such as the motors may not be spinning at the same speed or the wheels of the robot may differ slightly in terms of size and shape. To address this issue, a simple and easy-to-implement controller is used known as Proportional-Integral-Derivative controller, PID in short. It is a feedback controller that helps the robot to move in a straight line using constant velocity.

3.4.1 PID

Since the robot does not move straight when both motors are commanded the same value, a PID controller is used to make the robot move straight. As seen from fig 3.6, the setpoint

is the speed of the wheels in rpm. The process here is the rotation of wheels and encoders are responsible for reporting the speed of the rotating motors. This controller has three constants known as gains. These gains have to be tuned such that the robot moves straight. The PID controller is implemented in the ROS framework. The control board running ROS sends a command to teensy through the UART port and in turn teensy is responsible for commanding the motors at a given speed.

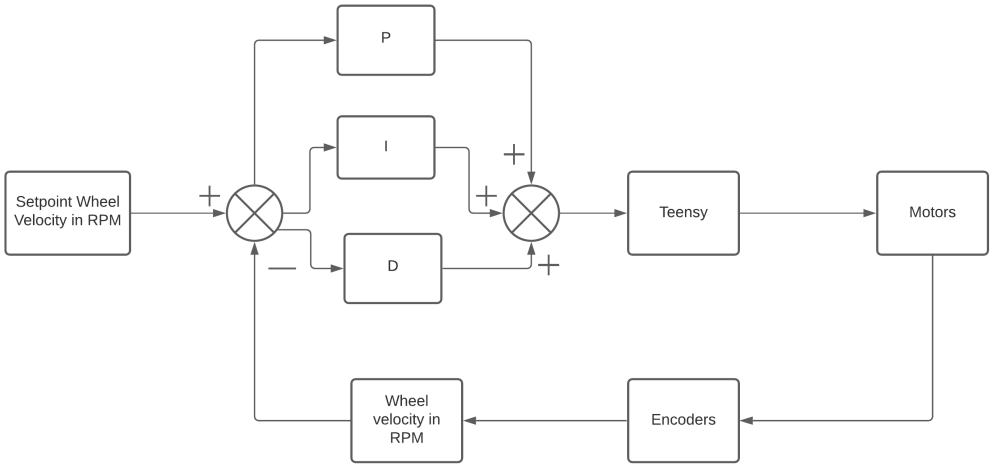

Figure 3.6: PID

PID Tuning

$$u(t) = K_p\, e(t) + K_i \int e(t)\, dt + K_d \frac{de(t)}{dt} \tag{3.7}$$

In the above equations, u(t) is the output of the PID equation. *"K_p"* refers to the proportional gain, *"K_i"* refers to the Integral gain, *"K_d"* refers to the differential gain.

The proportional gain describes "How fast the system should reach the setpoint" If the value is too less the system may never reach the setpoint. If it is too large it may overshoot beyond the setpoint and starts oscillating. The differential gain tries to smoothen out the

oscillations caused by proportional gain. The integral gain is responsible for minimizing the steady-state error that exists after tuning proportional and integral gain. Usually, the integral gain is assigned a small value, since it is scaling the accumulated error.

In this experiment, a PI controller is used to control the velocity of the wheels in RPM. As the proportional gain "K_p"was increased, the system approached closer towards the setpoint, but when *"K_p"* was set above 0.6, oscillations were high. Therefore, *"K_p"* was set to 0.6. The proportional controller is not able to reach the setpoint, but the oscillations are reduced and the system is more stable. To decrease the steady-state error, the integral gain is increased to 2.55. Derivative gain dampens the oscillations of the system. When proporional gain was set to 0.6, significant oscillations were not observed, Therefore *"K_d"* is set to 0. PI controller with the mentioned values makes the system attain the setpoint easily and is more stable.

3.5 Software

3.5.1 Robot Operating System

ROS short for Robot Operating system was introduced in 2003 by Stanford University and Willow Garage [31]. Before ROS, roboticists had to design and develop frameworks for robots, each specialized in a specific task. Due to this a lot of time and effort was invested in working on redundant things for making a robot functional. ROS introduced a framework that consisted of the necessary software tools which can be used to make any robot functional without any redundant work. It provides a sort of modular infrastructure that can be used by developers to test the ideas pertaining to robots. There are many community-developed packages for ROS that can be used by other roboticists in their research. ROS

provides powerful visualization tools that help in building software on robots.

As mentioned in the previous section, in this experiment, the robot is controlled by a Raspberry pi. this is a single board computer running ROS. ROS provides all the tools necessary to make the robot autonomously navigate by avoiding obstacles.

3.5.2 Unity

Unity is a 3D game engine that can render high-resolution textures, scenes with lighting and shadows and also has a sensor modeling feature. Unity editor has many features required to create a 3D game in its engine. Virtual reality headsets can be easily interfaced with Unity since it allows sensor modeling. It has many community-developed plugins that can be directly used in the unity environment from the asset store. In this experiment, a plugin called the Unity Multipurpose Avatar system is used to create an animated avatar.

The robot is equipped with an interactive avatar for communication. This animated avatar is simulated in a Game Engine called Unity. Unity provides a plugin called Unity multipurpose avatar or UMA for short. this plugin helps in creating different avatars and is tune-able to an individual's preference. ROS provides an API to talk to third-party applications called ROSbridge.

IBM's well-known speech synthesis platform is used for communication. IBM provides a Unity API to integrate IBM Watson into unity. IBM has a system, where a user can feed questions and answers. If the question asked to Watson is found in those queries, the corresponding answer is sent into its text to speech block.

The entire robot runs on Robot operating system, interacting with unity and IBM cloud. The complete system is as shown in fig 3.16.

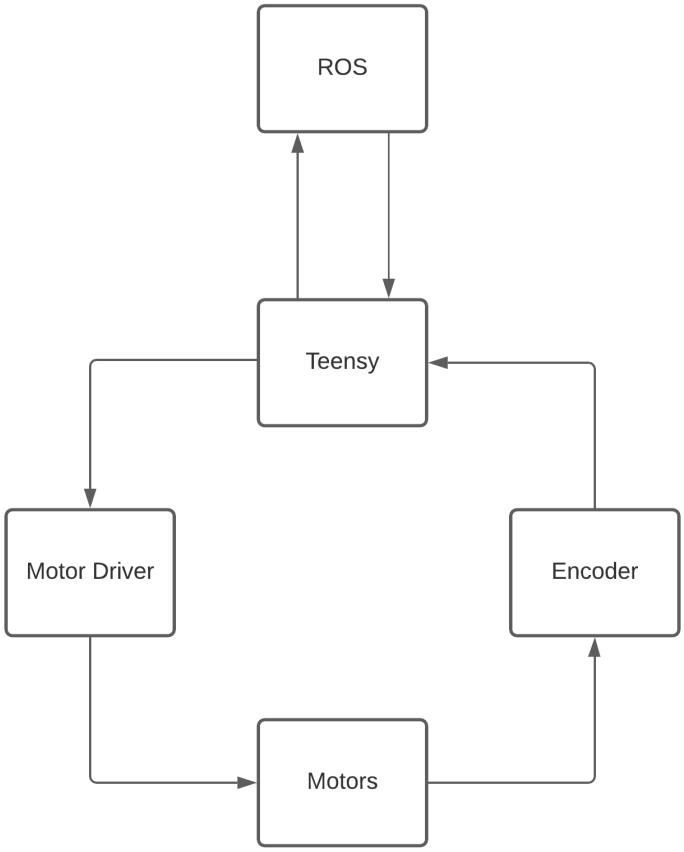

Figure 3.7: Lower Level Hardware for controlling Wheelchair

3.6 Processing and Fusion

So far the position and orientation are calculated solely based on the ticks provided by the encoder motors. The encoder is accurate when the robot moves forward in a straight line. However, when the robot rotates, there is a lot of drift and the position estimate provided by the odometry is not reliable. To address this issue another reliable sensor that provides the required orientation estimate of the robot is needed, which in this case is an IMU. The Real sense depth camera comes equipped with an IMU that provides linear acceleration and angular velocity. However, raw values provided by IMU cannot be used directly. In general,

IMU has a gyroscope and an accelerometer. The gyroscope provides the angular velocity and the accelerometer provides linear acceleration. The gyroscope gives an orientation estimate when the robot is moving and the accelerometer is good at estimating orientation when the robot is stationary. Therefore Sebastian Madgwick et al. [32], a filter is used known as a Madgwick filter to obtain a good estimation of the orientation of the sensor mounted on the robot.

In order to obtain close to accurate position and orientation of the robot, the values from these two sensors are fused using the Extended Kalman filter. The resultant position and orientation of the robot are obtained, which can then be further used by navigation algorithms to move the robot safely to its destination position on the map.

3.7 Navigation

Once the robot has been set up with the above requirements. It has to be interfaced with the Navigation Stack of ROS. This stack is responsible for making any mobile robot autonomous. However, before implementing the navigation stack. There is a need to build a map of an environment. The navigation stack works effectively when it has a prebuilt map.

3.7.1 Mapping

When it comes to building a map of an environment. There are many algorithms that are developed over the past years such as RTAB map, occupancy grid map, octomap to name a few. In this experiment, the map to be built is called the occupancy grid map. Occupancy grid map uses inverse sensor model [33] to build a map, where the robot uses its current position and a laser scanner to estimate the placement of obstacles. This map is a grid

map, as the name suggests, the entire environment is perceived as a 2D grid world to the robot, with each cell in the grid containing information whether it is being occupied by an obstacle or not. The probability of a cell being occupied is considered to be one [p(the cell has obstacle) = 1], the probability of a cell not containing an obstacle is zero [p(the cell has no obstacle) = 0], and the unexplored grid cells have values -1. This way a map is being constructed. ROS makes this easy to implement by providing the necessary tools to generate such a map and visualize the process of map building.

This experiment chooses the Occupancy grid map (OGM) over the other maps due to various reasons. OGM is not computationally heavy and it requires a 2D scan of the environment, it requires the position of the robot to be accurate. if the robot's position is not accurate, this will lead to numerous problems which we will discuss further in another section. However, the mapping package that is available in ROS tries to localize the robot using a laser scan matcher. The map created by the robot is divided into two subsets called the Global Cost map and Local Cost map. These cost maps have to be tuned for the navigation stack to control the robot in the map smoothly.

3.7.2 Cost map

As explained above, local and global cost maps are required for the smooth navigation of the robot in the given environment. The cost map, as the name suggests holds the information of the cost of each cell in the grid created while mapping. These cost maps have an inflation radius. Inflation is a bubble around the obstacle that signifies a gradual increase in the cost of the cell as the robot approaches the obstacle. These cost maps have different layers. Each layer has its own significance and inflation radius. These layers are

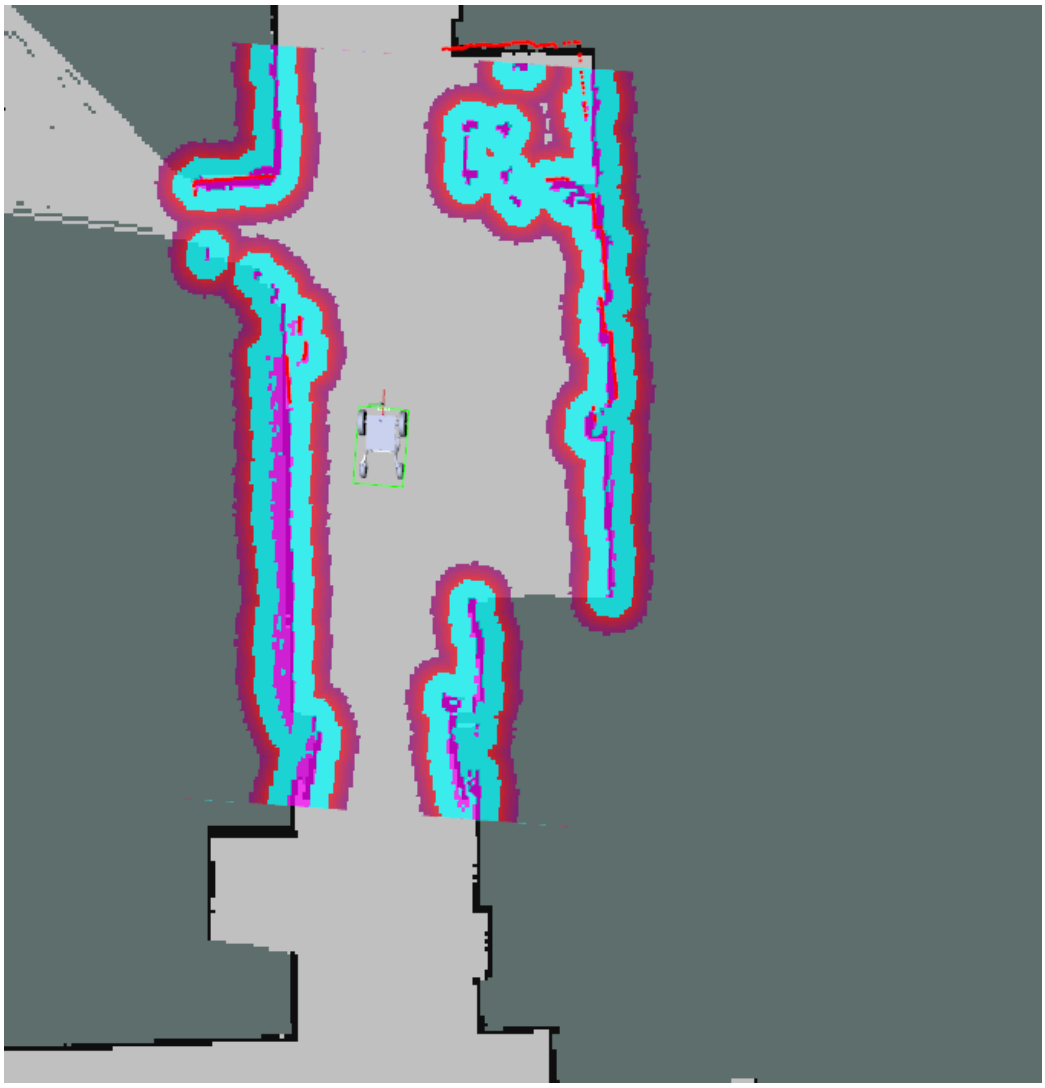

Figure 3.8: Local Cost map

as follows:

- **Static layer:**

  This layer holds information about the obstacles in the static layer. example borders of the map. The inflation radius of the static layer can be different than other layers. This layer has to be included in the global cost map, as the global planner can use this information to plan the path. The local cost map also includes this layer as to not let

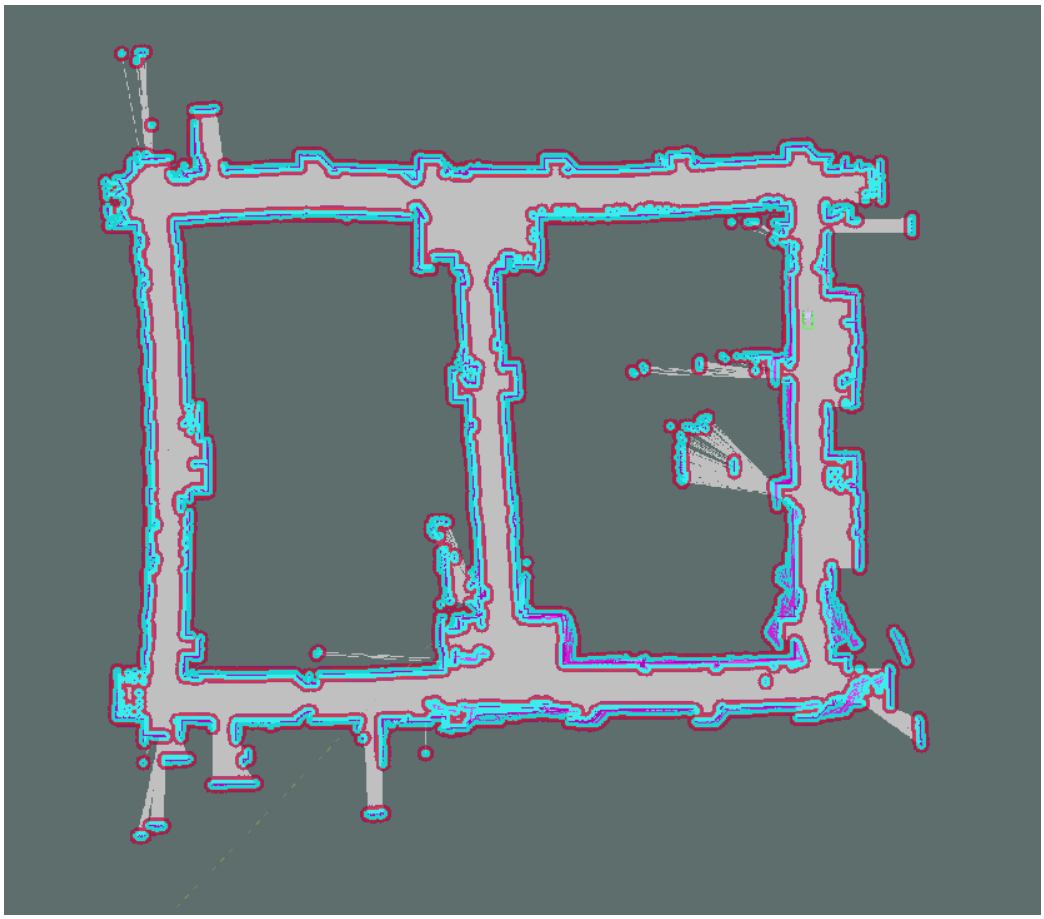

Figure 3.9: Global Cost map

the robot crash the static obstacles.

- **obstacle layer:**

  This layer holds information about obstacles that are not present in the static layer. The obstacle layer is used by the local cost map. The local planner is responsible for planning the path avoiding these obstacles by sending corresponding control commands to the robot. This layer also contains an inflation radius signifying the nearest distance the robot can travel around this obstacle.

- **Inflation layer:**

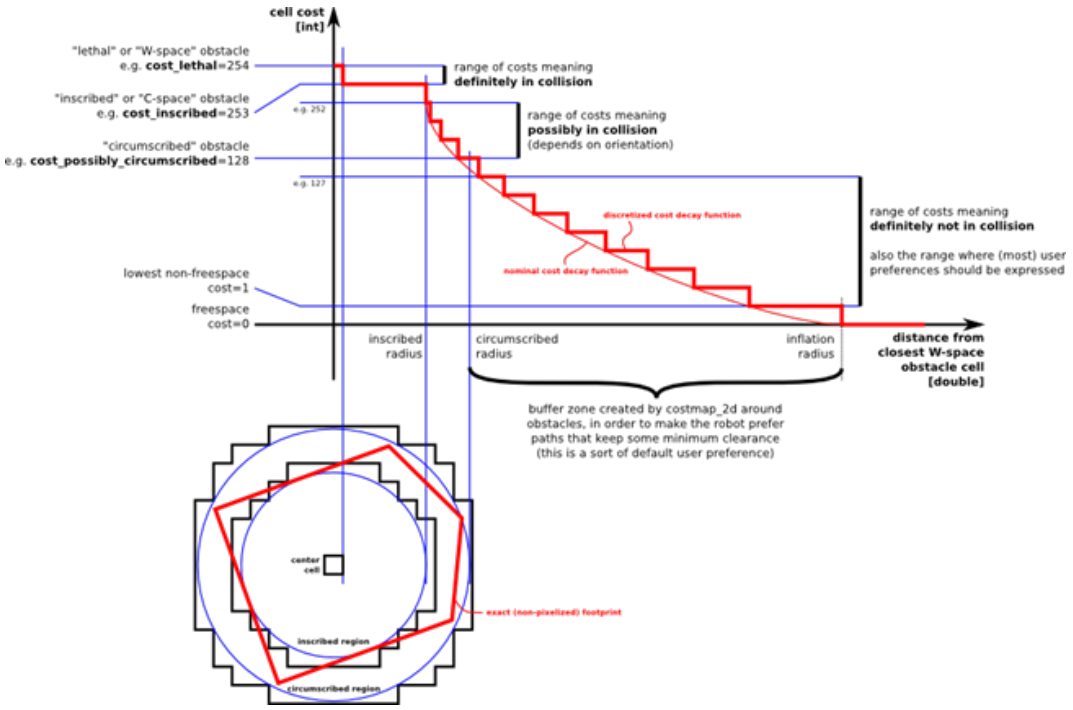

Figure 3.10: Costmap 2D [39]

This layer holds information about different levels of cost in the gradually increasing cost of inflation radius as the robot approaches the obstacle.

The figure 3.10 depicts the different costs of the inflation layer, Lethal cost means that the robot is in collision with the obstacle.

Inscribed cost also means that the robot is in collision with an obstacle. Possibly circumscribed cost defines the possibility of collision of the robot with the obstacle depends on the robot's orientation.

Freespace cost means that the robot can maneuver that area without collision.

The costmap2D has many more layers which are not discussed in this section as they are not used in this experiment.

After tuning the cost maps along with local and global planner, the robot is now ready to

navigate the environment by itself. ROS provides an API class that can be used to command the move base by providing goal points to its server.

After building a map, the Navigation stack can use this map to maneuver the robot in an environment. The navigation stack requires a prebuilt map and the ability to send directional commands to a mobile robot. This navigation stack has three main algorithms and its parameter has to be set up depending on the mobile robot. The algorithms are as follows:

3.7.3 Adaptive Monte Carlo Localization

There are many localization techniques available in ROS such as AMCL, EKF localization, and so on. AMCL stands for Adaptive Monte Carlo localization. It is a probabilistic localization system for robot maneuvering in a 2D environment [34]. Here The robot is interpreted as a particle. These particles are randomly spread in the given map. each particle has a specific weight. Depending on those weights the particles are sampled from the distribution. AMCL matches the LaserScan of the particles to the borders of the given map. If the scan matches the map, then those particles are sampled from distribution as they have a higher weight than all other particles. AMCL depends on the robot's odometry sources and LaserScan measurements. The figure 3.11 depicts the localization of the robot using AMCL while navigating in the 3rd floor KGCOE. In this experiment, Extended Kalman filter is used for correction of drift in encoder source. This fused odometry is an input for AMCL localization.

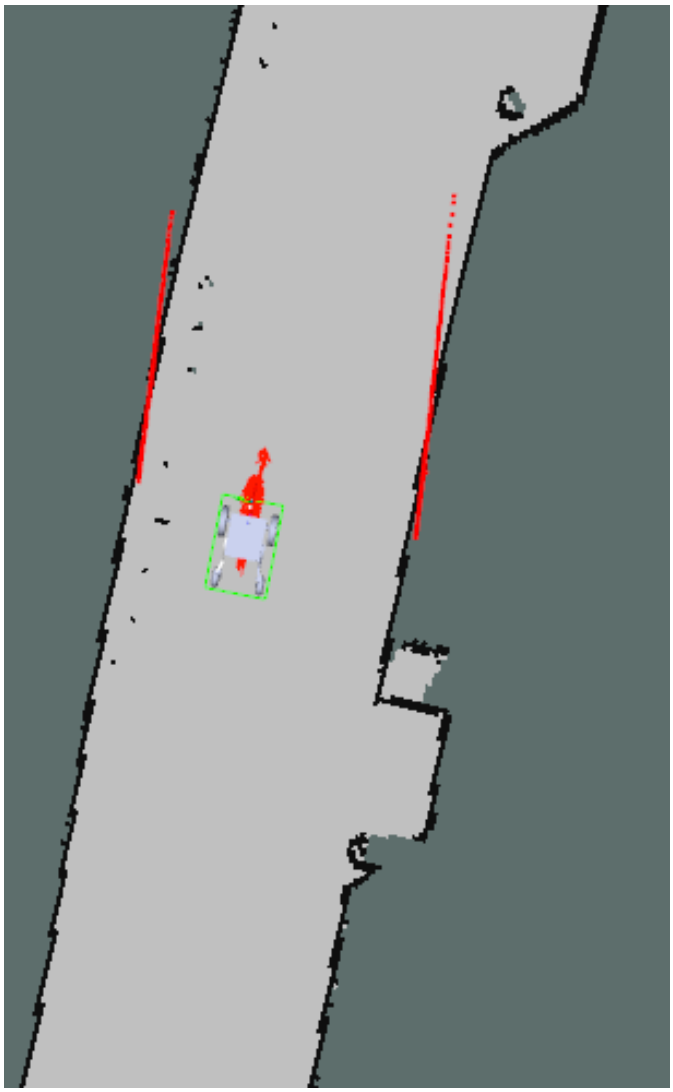

Figure 3.11: Particle filters localizing the robot

3.7.4 Move Base

Move Base as the name suggests is responsible for moving robots from one point to another in a known map. The map server loads the map so algorithms in the move base can access it. Move base has two important helper algorithms, which are Global planner and Local planner. The Global planner is responsible for creating a goal from one point to another on the map while considering the borders of the map and static obstacles scattered all

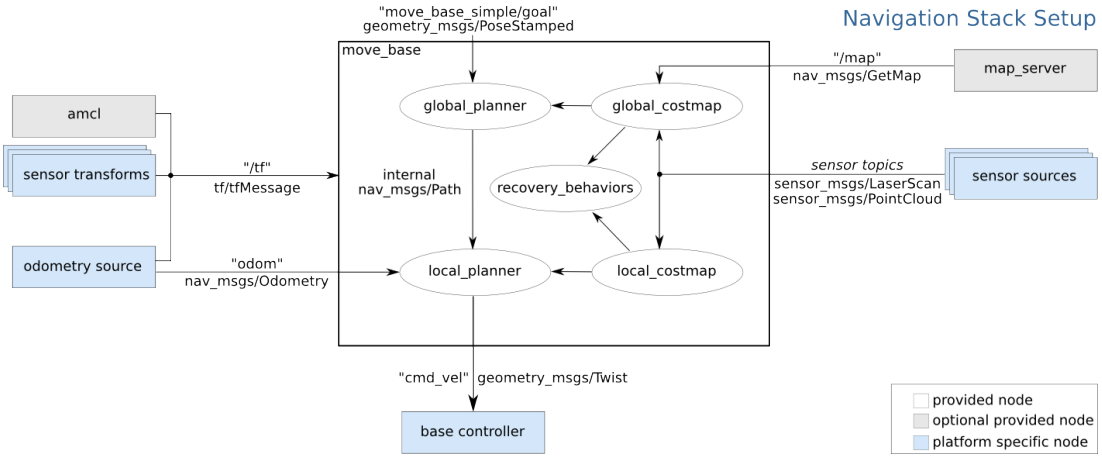

Figure 3.12: Navigation Stack in ROS [38]

around the map. The local planner is responsible for commanding the robot to follow the path created by the global planner and avoiding any dynamic obstacles that are not on the prebuilt map and help in reaching the destination safely. There are many algorithms that can be chosen. In this experiment, A* algorithm is used as a Global planner. The local planner used here is the Dynamic-Window Approach planner, which is an effective algorithm for making the robot autonomous. The global planner is responsible for creating a path between the robot's start point and its destination on the occupancy grid map. The map loaded has two configurations, they are Global CostMap and Local CostMap. Global planner uses the global cost map to trace out the path between the start and endpoint. The local planner is responsible for maneuvering the robot in the path provided by the global planner by avoiding obstacles that are in the way of the robot. It does so by using the information provided by Local CostMap. The global cost map consists of the map borders, local cost map consists of the obstacles that are not present on the map while mapping. Fig. 3.12 shows the structure of navigation stack, where move base takes its input from

odometry source, map server and frame transforms to drive the robot autonomously in a known map.

3.7.5 April Tags

This experiment also consists of visual fiducial markers as artificial landmarks in the environment. April Tags are a kind of visual fiducial marker that consists of a black background with white foreground with a specific pattern. Due to the black border of the April Tags, it is easy to detect using computer vision techniques under various conditions like poor lighting, different orientations, etc. April tags look like a QR code. However, a QR code holds around 3Kb of information whereas April Tag can hold only 7 to 12 bits of data. This is the important feature of the April tag since it has less payload and is detected at far distances without any difficulties. In this experiment, Tag 36h11 April tags are used. This is a standard April tags family. ROS provides support for detecting April Tags and visualizing them in a tool called RQT image view. This ROS package requires the size of the April tag and the tag family used for this experiment to detect the tags and finally camera stream is provided as an input.

3.8 System Integration

3.8.1 Unity and ROS integration

ROS exposes its API for interfacing with any software. ROS bridge is used for communication between ROS and unity. ROS# is a powerful set of software libraries used for communication between ROS and .net applications, specifically unity through WebSockets using the JSON framework.

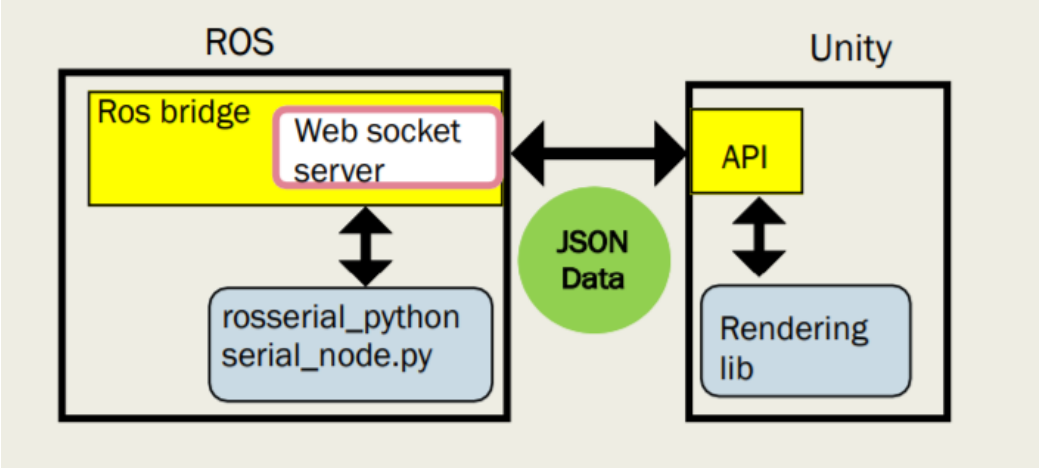

Figure 3.13: ROS and unity interaction with ROS bridge

Unity can parse the URDF file available in ROS and can simulate the model in its editor with the necessary mesh files. A lot of experiments are conducted on virtual reality and robotics, where the interface between ROS and Unity plays a vital role.

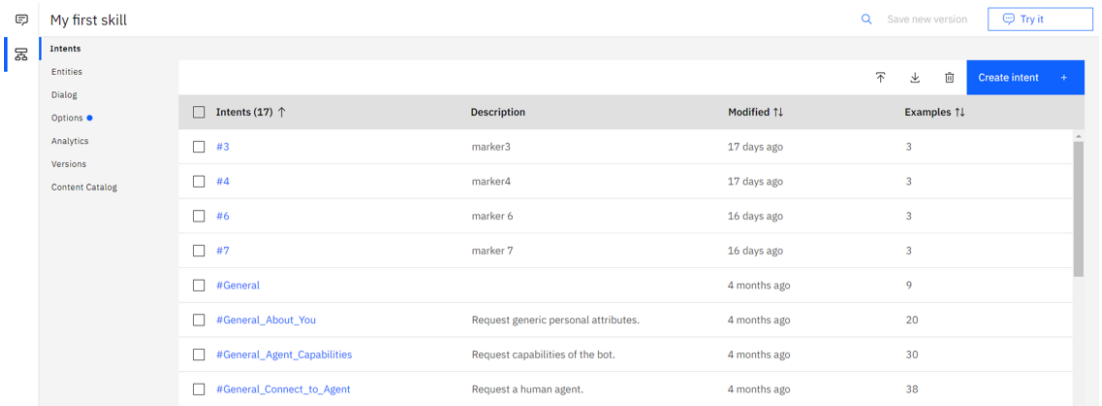

Figure 3.14: IBM Watson dialog system

3.8.2 Unity and IBM Watson Integration

IBM Watson exposes its API in C# programming language. This API can be used in unity to make these systems communicate with each other. The basic criteria here is IBM Watson requires access to the microphone and speaker of the control board. Once the permissions

are provided, IBM Watson can interact with users through the Unity game engine. Figure 3.14 shows the dialog system. The speech recognition converts speech to text, which is then processed in Watson's dialog system. The dialog system generates an answer for the question asked to the Watson, which is then sent to text to speech. the text to speech system is responsible for playing the text as an audio.

3.8.3 Unity Avatar System

Figure 3.15: Animated Avatar [41]

Unity comes equipped with Asset Store. An avatar system called Unity Multipurpose avatar and a lip syncing plugin known as Salsa plugin is available on the asset store. This Avatar system has the capability of lip-syncing and animation. The avatar can be tweaked to the user's taste. In this system, A female avatar is chosen and is displayed on the screen.

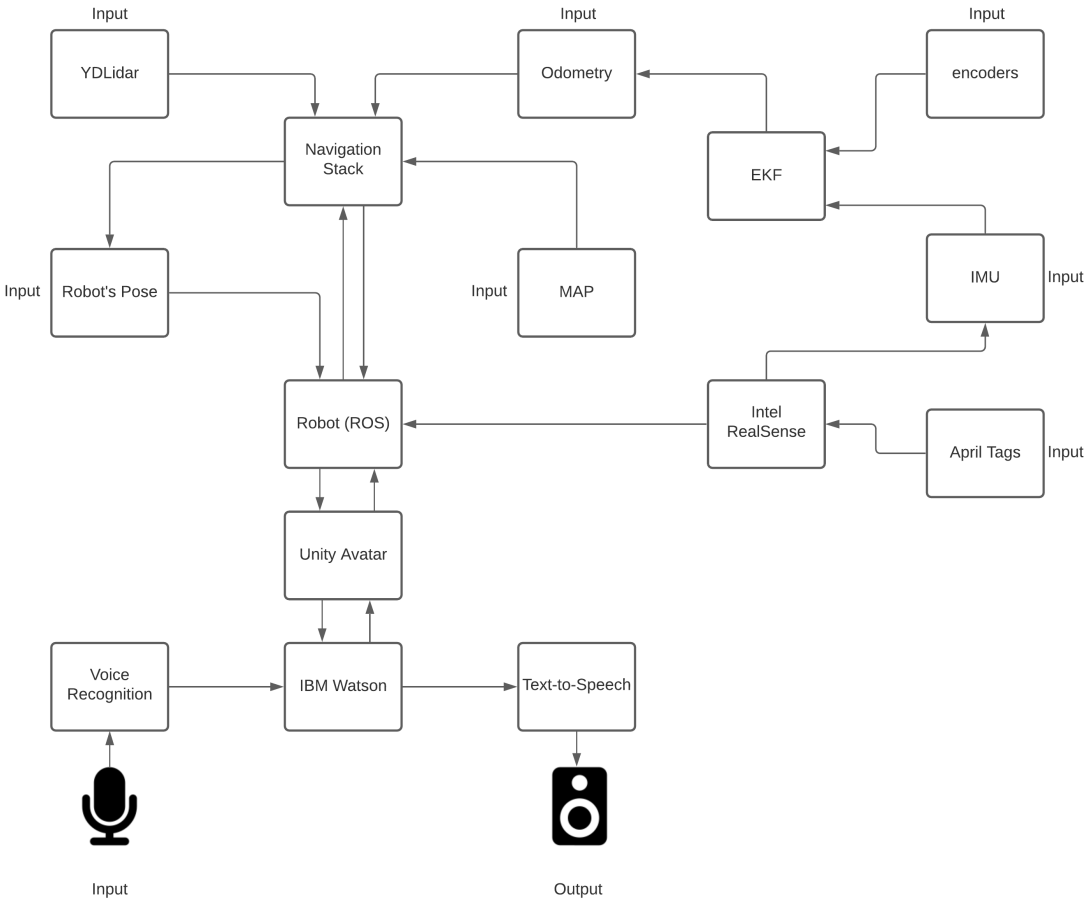

Figure 3.16: Complete System Block Diagram

In fig 3.16, each block represents a different part of the system. The experiment here is a tour-guide robot. The robot should be able to move from its starting point to its destination avoiding obstacles in its way and It should also be able to interact with humans. To achieve this, each room is stuck with April Tags. each tag has a unique code that can be deciphered from the vision system of the robot. These tags are 7x7 large and around 1024 unique markers can be generated. These tags are stuck to the laboratories in the environment. The robot can be made to maneuver near the tags. The robot is equipped with a vision system

capable of detecting the markers. Once the tags are detected, the information about the marker id is sent to IBM Watson.

IBM Watson has a dialogue system. if the specific tag id exists in the dialogue system, the corresponding dialogue is submitted to the text to speech conversion system. This system can access the speaker where the speech is being played. Once the speech is finished, the unity pings the ROS framework about its current status, the robot starts moving to the next goal point. This keeps repeating until all the tags have been visited in the environment.

The odometry source of the robot comes from two sources as shown in fig 3.16. Intel real sense has an IMU built-in, which provides information about linear acceleration and angular velocity. The encoders attached to the motors provide information about the position, orientation, and velocity of the robot. The data from these two sensors are passed to Extended Kalman Filter, which then gives a prediction of the robot's pose with position and orientation. This fused odometry is then fed to the SLAM block for mapping and navigation.

The navigation stack is responsible for the autonomous navigation of the robot. The data from the perception sensors are fed to the navigation stack or move base. The localization node AMCL, Global planner A* algorithm, local planner TEB planner work together to maneuver the robot in the given environment.

Chapter 4

Experiments

4.1 Hardware

The initial setup consisted of a raspberry pi model 3B+ as the heart of the wheelchair, which acted as a ROS master. This Single board computer consisted of 1GB Ram and a quadcore processor. However, It was not able to simultaneously run the algorithms and communicate with the computer, which led to the robot wavering too much while moving from point to point. The processor was powerful but it was bottlenecked by its Ram Capacity. Since the onboard GPU was not powerful enough, it could not run any visualization tools available in ROS. Due to which all the visualization tools were run on a computer.

The raspberry pi was connected to a router. Any computer that can connect to this network would be treated as a ROS slave. ROSCORE would be run on the Single Board Computer (SBC), which was required for network communications between multiple machines. In order to validate the process of the SBC's underperformance, It was replaced with a laptop. The laptop was much powerful in terms of Processor capability, GPU, and most importantly RAM. After replacement, the robot had less wavering and could travel smoothly from point to point on the map. The system was also tested on Nvidia's Jetson Nano, which is a powerful alternative to Raspberry pi with much more ram and GPU. This

SBC could also run visualization tools as it had a more powerful Nvidia GPU.

4.2 Mapping

The experiment was first conducted in a space where the robot had more freedom to choose its path. The Mapping of such an environment does not require a lot of work. So a simple odometry source like Encoder was enough to fulfill mapping capabilities. The gmapping relies heavily on accurate odometry sources and laser scans. Therefore, it is not recommended to use gmapping if the odometry source is not accurate. The robot in this experiment has a very good CUI Encoder which provides around 118,784 ticks per revolution of the wheel. However, The encoder is useful when driving in a straight line and has issues during the turning of the robot, where there is a lot of drift.

In the figure 4.1, The map is created by using a laser scan and odometry of the robot. When the robot rotates in place there exists drift in its encoder source which in turn leads to the robot losing its position and creating a map that overlaps on top of each other making this map invalid to reuse for navigation stack.

Initially there was no need for drift correction in this experiment as AMCL could navigate the environment. However, when the robot has to map different places that consist of a lot of corridors, it is important that it creates the map almost accurately. If the robot has to make too many turns, it is very likely that the robot would not be able to create a good map due to the drift in the encoder.

In the map shown in the fig 4.2 and 4.3, even though the map looks similar to a rectangle and portrays the 3rd floor, this map has a key error that makes it unfit for navigation. Since the 3rd floor of KGCOE consists of corridors, The corridor reading is the same for at least

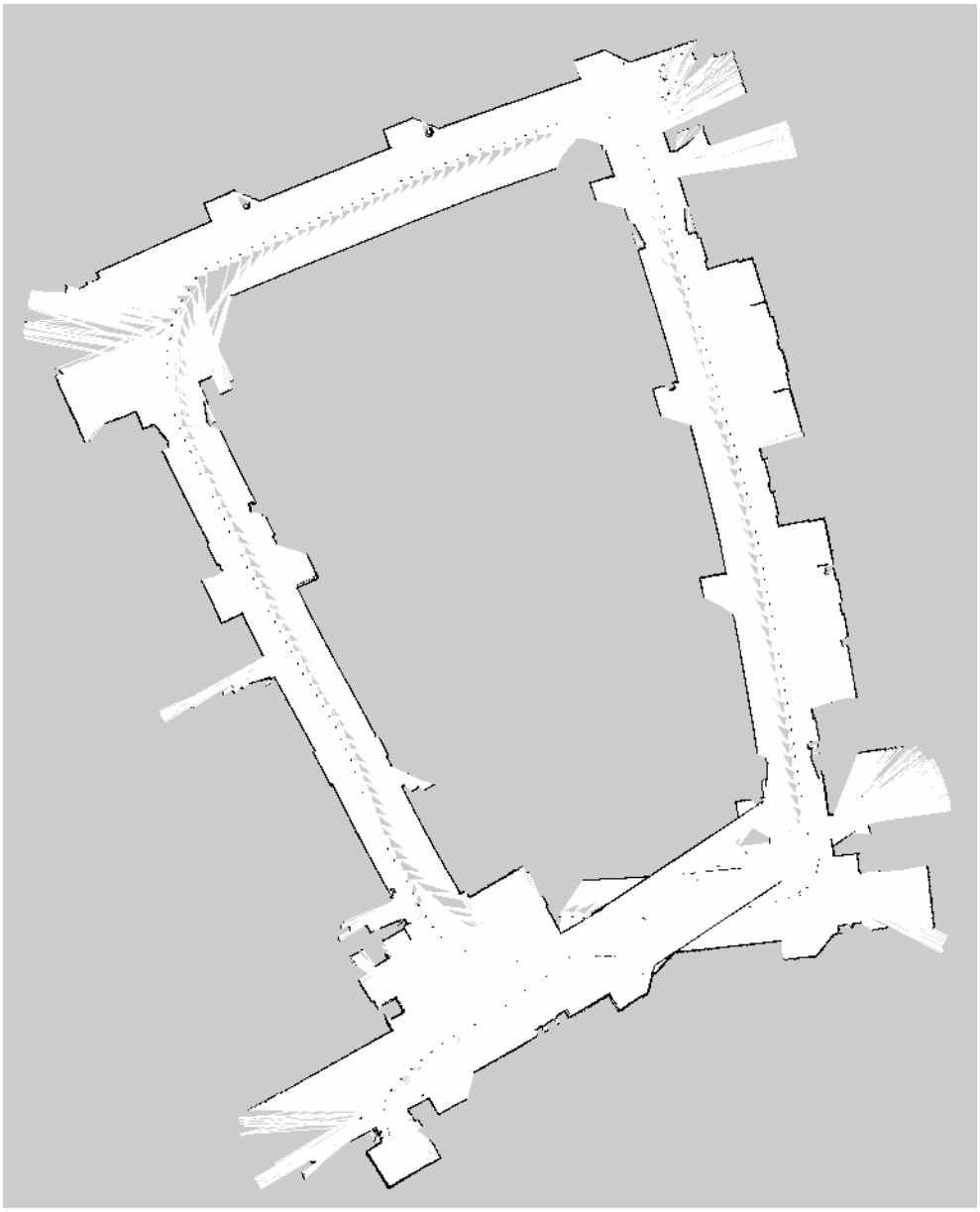

Figure 4.1: map generated with encoder

2-3m, which makes the algorithm believe that it has not moved and is in the same place as it first registered the scan of the corridor. This makes some pathways in this map smaller, which confuses the localization node and the robot gets stuck at some point.

In this experiment 3rd floor of KGCOE, RIT is mapped with the encoder to check the

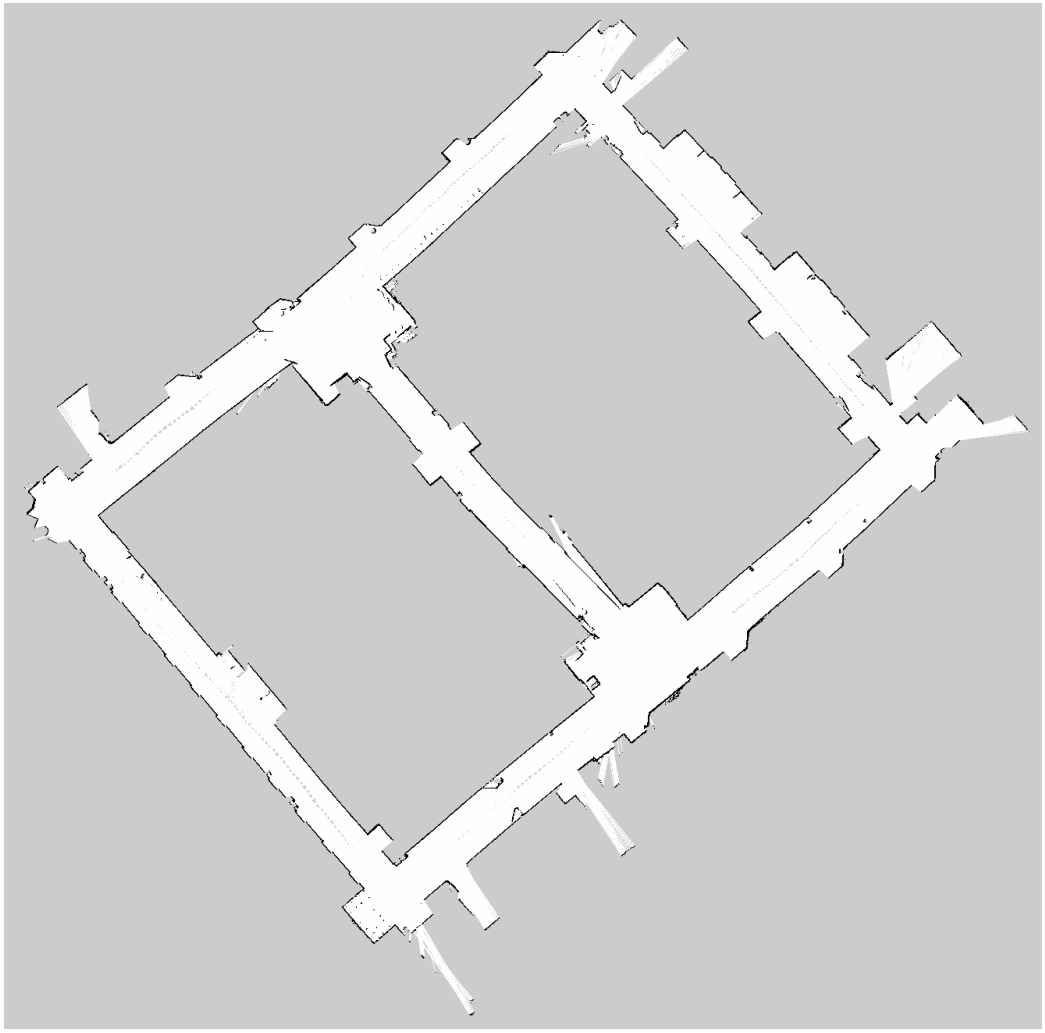

Figure 4.2: Map generated with encoder and IMU EKF fusion

quality of the map generated by the robot. The corridor sections were overlapped on top of each other as the robot lost its position due to the drift in encoders. To overcome this issue of drift in the odometry source and create a good map, an Inertial Measurement Unit sensor is used as another odometry source. The Intel real sense D435i comes with an IMU built-in. This IMU was used to correct the drift of the robot. IMU's linear acceleration in forward direction, which is in the x-direction, its angular velocity in z-axis which is yaw, and its orientation values are fused with encoder's forward velocity, angular velocity, and

orientation using Extended Kalman filter package available in ROS to create an Odometry source that has an almost precise position of the robot in the real world.

This fused odometry source was used in mapping the places with corridors. The map created by this new odometry source was better than the map created by the encoder. However, it had its own problems. Even though the map created could be used for autonomous navigation, the robot could not navigate some parts of the created map, which had overlapped to a certain extent. Those overlap portions of the map were because gmapping could not accurately determine the loops present in the environment. Gmapping would detect the loops and would try to correct the map but the correction was not accurate enough, which would lead to the overlapping of this portion of the map. This would lead the AMCL localization algorithm confused in the overlap sections of the map. Hence, the robot would not be able to traverse the path correctly.

Figure 4.3 is another example of a map that cannot be used for navigation. This map suffers at the point when mapping is closed and the map gets overlapped on each other. This is known as loop closure. If the loop is not properly closed, the map becomes unfit for use in navigation.

Figure 4.4 depicts the map generated using karto SLAM [45]. The odometry source for this SLAM is provided by EKF fusion of encoders and IMU. The map constructed with this technique overlapped and the required loop closure was not provided. Hence, it was discarded.

To overcome the above-presented problem. A map was created from the floor plan of the environment as described in this algorithm as shown in [12]. This algorithm uses the

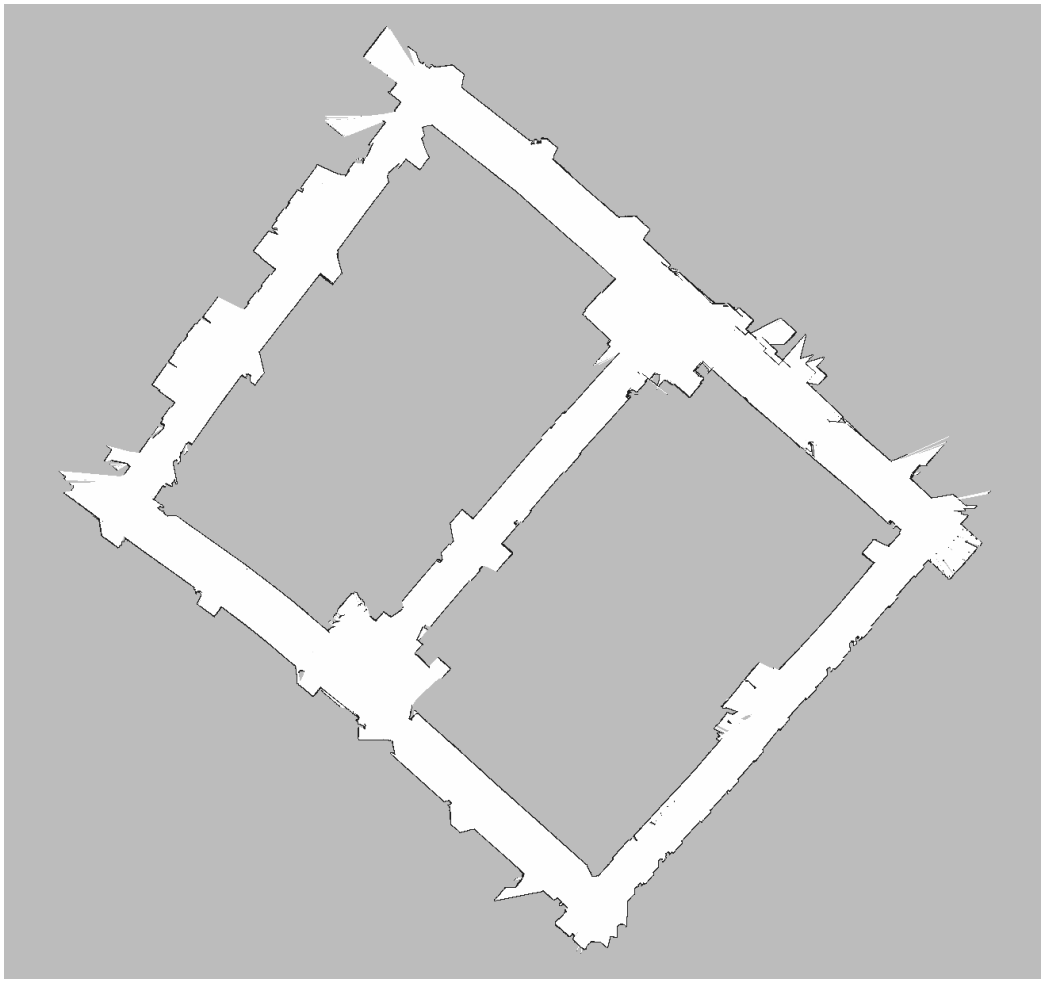

Figure 4.3: Map generated with UKF fusion

floor plan and it also uses two points on the x-axis and 2 points on the y-axis. It requires the distance between those points in the real world. Using the information provided in the floor plan and the distance data from the real world, the algorithm scales the floor plan and an occupancy grid map is created that can be visualized in ROS.

The map shown in fig 4.5 is used for navigation. This map is not created by the robot and its onboard sensors. This map is obtained from the floor plan of the 3rd floor of KGCOE. The floor plan is first converted into a binary grayscale image and then later fed into an

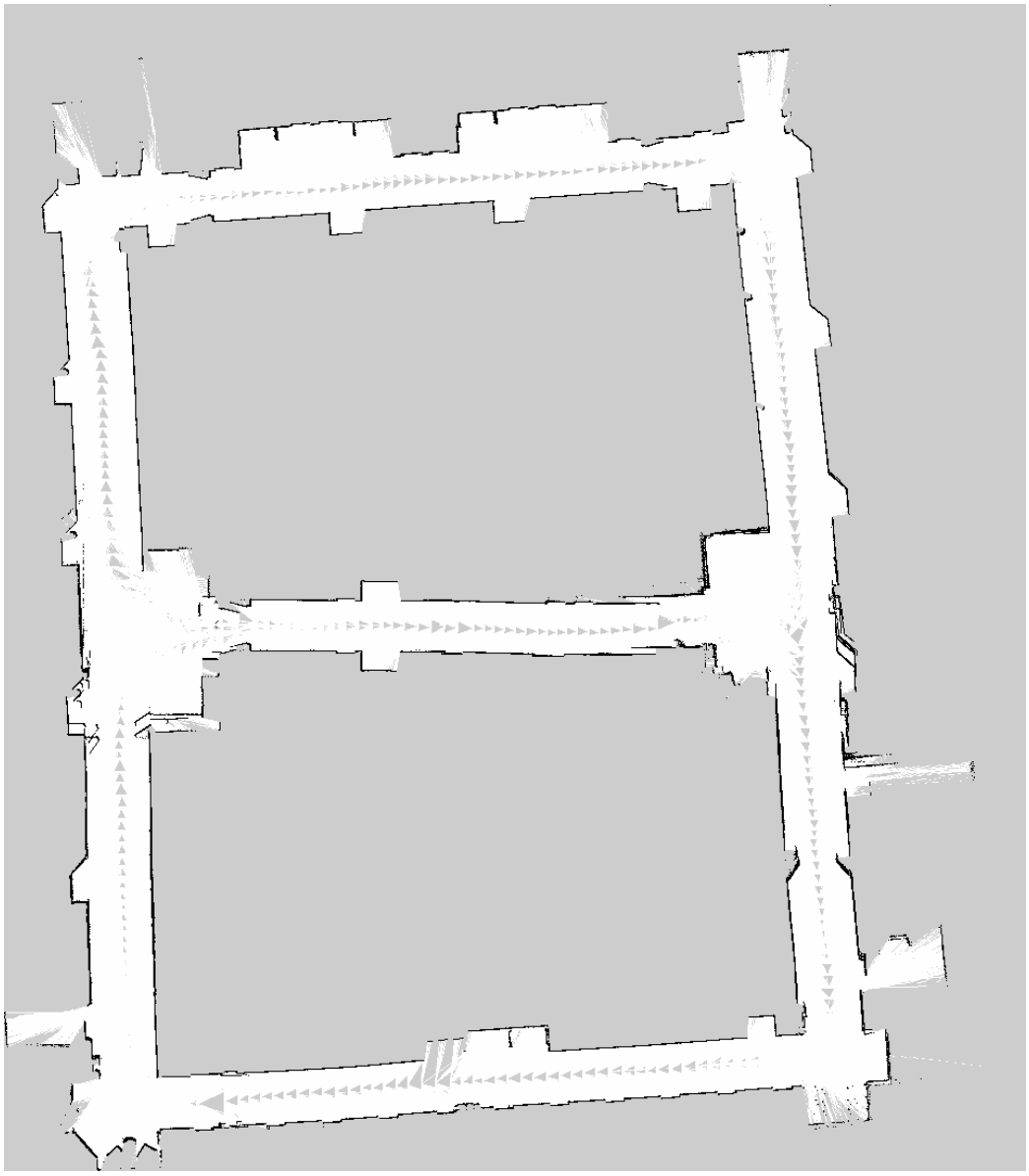

Figure 4.4: Map generated using Karto Slam

algorithm that converts grayscale into occupancy grid mapping, which can be used for navigation of the robot. Each cell in the occupancy grid map has a resolution of 0.05 meters.

Another mapping technique called Hector Mapping was run on the robot. However, Hector mapping does not use any odometry source for estimating the position of the robot.

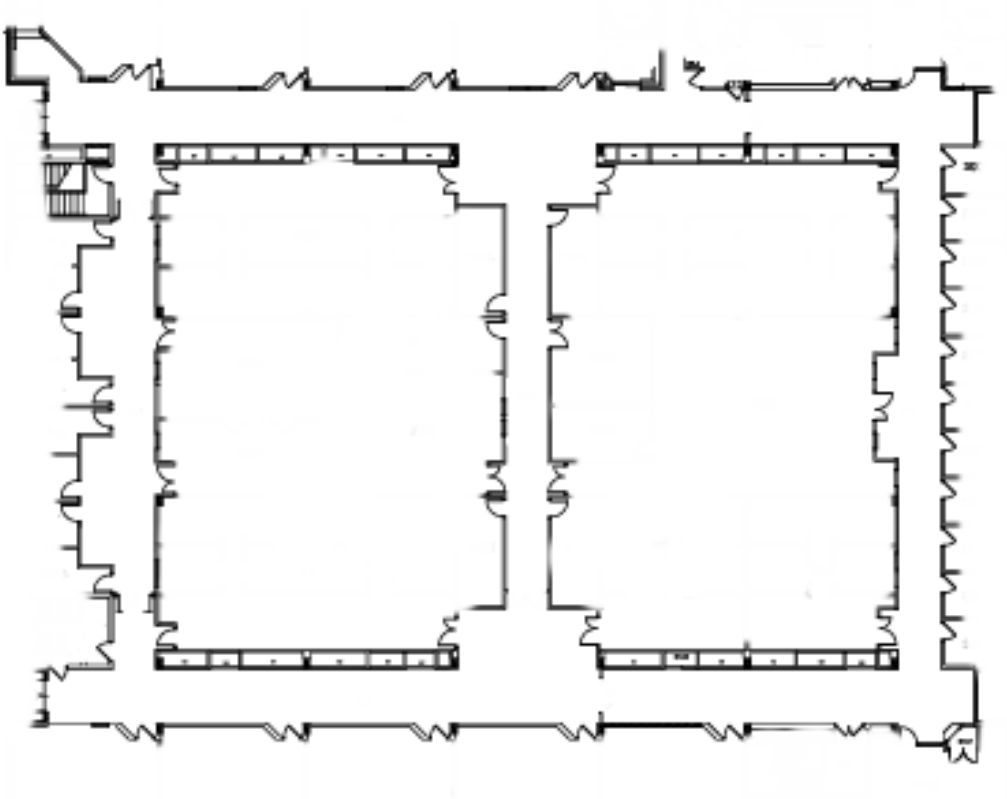

Figure 4.5: Map generated from floor plan

This mapping technique uses the Laser scan just to estimate its position and map the environment. However, since the corridors of the environment do not vary much, this mapping technique fails to localize the robot accurately. Hector mapping can be used in an environment which has a lot of features and when no odometry is required.

4.3 Global planners

The initial setup consisted of Dijkstra as the global planner. However, the heuristics of Dijkstra is the distance from the robot's current position to the start point. This would make the robot move very close to the walls and took a longer route instead of the shorter one. To overcome this problem, the A* algorithm is used. The heuristics of the A* algorithm is

the sum of the distance from the current position of the robot to the starting point and the distance from robot's

4.4 Local Planners

Many local planners are available in ROS. The default local planner in ROS is the Dynamic-Window Approach planner. This planner works well in a given map with static obstacles. If The local cost map has local obstacles, this planner stops as soon as it encounters an obstacle and this planner takes time to calculate trajectory score from its sampled velocities (v, w) and then moves past avoiding the obstacle in its vicinity. However, it sometimes fails to produce a path.

TEB local planner works great when there is an obstacle in the local cost map that does not exist in the static map. However, this planner is suitable for car-like robots. It does not take advantage of the differential drive turning mechanism. This planner requires more space in the environment to avoid obstacles.

EBand local planner is also suitable for dynamic obstacles. However, in this experiment, this planner made the robot waver a lot and the motion of the robot wasn't smooth. So it was discarded.

Chapter 5

Results

The first step towards making the robot autonomous with a prebuilt map is to observe the planners capability in maneuvering the robot autonomously in the presence of local obstacles in the costmap.

Local Planners	**number of Trials**	**successful trials**
Time Elastic Band (TEB)	9	8
Dyanimc Window Approach (DWA)	9	6

Table 5.1: Dynamic obstacle test

the table 5.1 represents the accuracy of a planner in avoiding obstacles in 9 trials. In all the trials that were conducted, the robot was autonomously navigating in the given map, while doing so, an obstacle is introduced in its path which would appear as a local obstacle on a local costmap, which means that the local planner is responsible for avoiding the obstacle. The TEB planner was successful in avoiding obstacles in more trials than the DWA planner. Hence TEB is chosen as the local planner in this experiment.

Once the planner has been selected, it is important to test out the planner in the environment. In this experiment, the autonomous capability of the robot is tested on the 3rd floor Kate Gleason College of Engineering (KGCOE), RIT. Some of the results are as shown below:

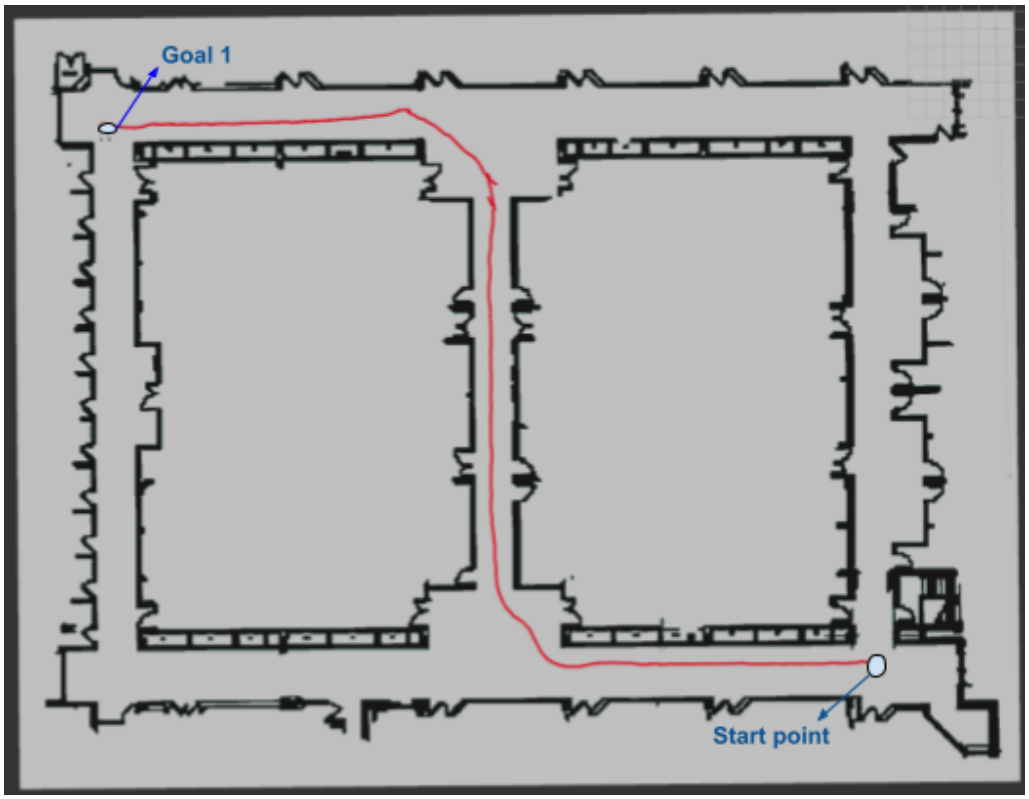

Figure 5.1: the trajectory followed by the robot to reach Goal 1

In the fig. 5.1, the start point is the starting point of the robot, and goal 1 is the destination. in this figure, the robot travels the longest distance on the map from one corner to the diagonally opposite corner. there are no sub-goals while traversing the map. As seen from the figure, the red line depicts the trajectory of the robot while traveling along the three corridors of the environment and reaching its goal location.

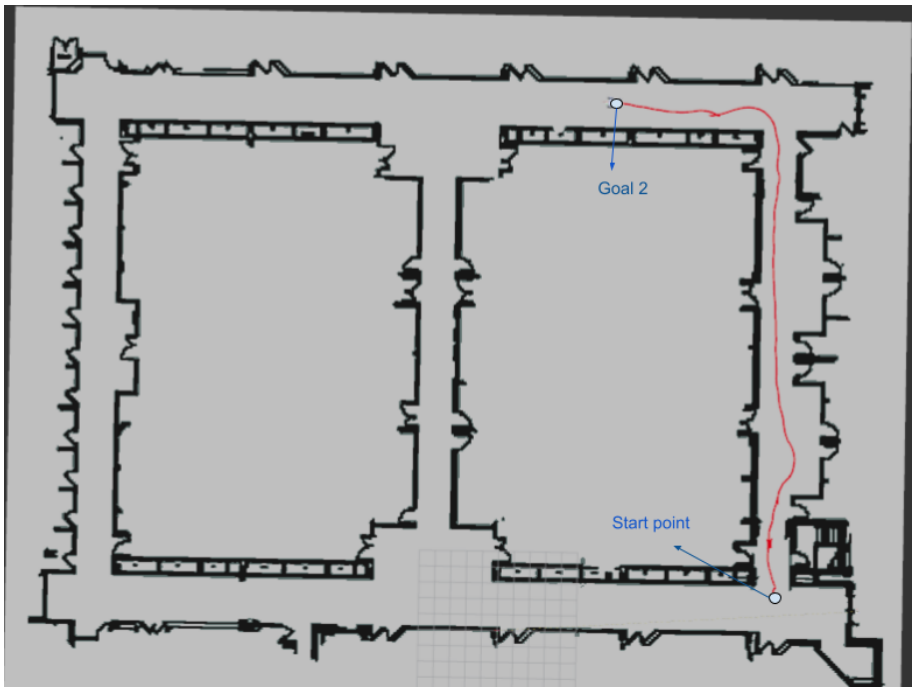

Figure 5.2: the trajectory followed by the robot to reach Goal 2

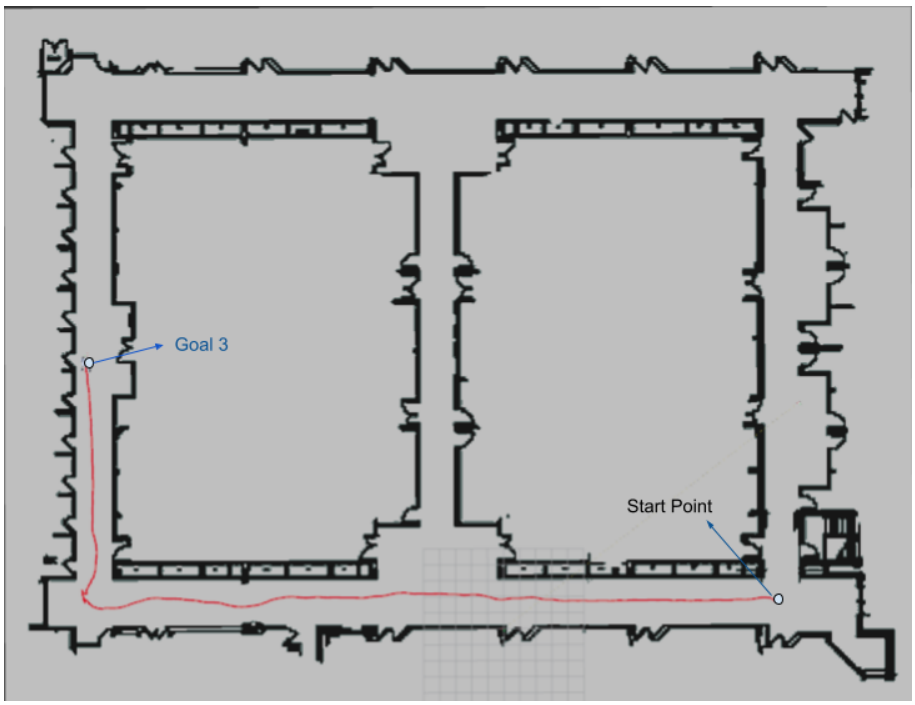

Figure 5.3: the trajectory followed by the robot to reach Goal 3

Figures 5.2, 5.3 represent the trajectory of the robot maneuvering in the remaining corridors of the environment that was not depicted in fig 5.1.

In figures 5.1, 5.2 and 5.3, the robot starts at the same start point and is able to traverse three different goal points. This shows that the robot can maneuver all part of the map autonomously.

Goal	**Distance Travelled along X-axis in meters (m)**	**Distance Travelled along Y-axis in meters (m)**	**Time taken in seconds (s)**
Goal 1	42.778	31.055	419.448
Goal 2	10.831	30.813	274.789
Goal 3	43.045	15.15	323.658

Table 5.2: Distance travelled and Time taken to reach goals

Table 5.2 represents the distance travelled and time taken by the robot to reach goals as shown in fig 5.1, 5.2 and 5.3. As per the table, the robot takes 419 seconds to reach goal 1 since it has to travel long distance. Goal 2 is near to the start point of the robot compared to goal 3. Hence, the robot reaches goal 2 faster than goal 3.

The final system consisted of a robot navigating autonomously in the indoor environment and an avatar for human-robot interaction. The robot needs a prebuilt map to navigate autonomously. The April tags are stuck to the doors of the laboratories and their locations are known with respect to the map frame. The navigation stack is provided with the points which are closer to April tags. The Intel real sense camera can detect the April tag effectively and relay information to Unity which would trigger a dialog in IBM Watson explaining the people about the experiments that are going inside these labs. The entire system is not as it is expected to be. There are issues that need to be addressed. The local planner's parameters had to be tuned to match the environment. Local planners like DWA, TEB,

Eband were tested and finally TEB planner was chosen since this planner is smooth and executes trajectory avoiding obstacles. Care should be taken to keep a minimum distance of 3-4 ft when giving tours so that the robot wouldn't crash when it is driving backwards.

There are a number of situations such as the overlapped portions of the map where the localization algorithm fails to localize the robot and the local planner does not provide effective control signals to the robot which might lead to the collision of the robot with the obstacles. The 2D lidar is mounted in front of the robot. The lidar has a range of -180° to +180° which means it can sweep the entire area surrounding it. However, the range of the lidar is limited to -90° to +90° in this experiment. The lidar can interpret the obstacles that are in front of it and disregards everything else that is behind it. Due to this limitation, there is a chance that the robot might crash a moving person or obstacle while the robot is moving backward.

If the path on the map is completely blocked by dynamic obstacles, the robot tries to move forward and backward calculating different paths it can take to avoid obstacles. the local obstacles appear on the local costmap. If the obstacle stays in front of the robot for a long time, the cell cost of the obstacle keeps on increasing which takes time for the algorithm to clear the map when the obstacle is moved out of the way.

The circuitry of the robot is not fully enclosed. In case of the spillage of liquid on robot might cause the circuitry to fail and the robot might not function.

The Avatar system is created with Unity 3D game engine which is power-hungry. Running Unity along with ROS on a laptop is not an effective idea as the entire tour cannot be covered with the available battery capacity of the laptop. In this experiment, to test out the

algorithms and functionality of the system integration, all the applications were running on a single laptop. As specified in the previous sections, Nvidia Jetson Nano is available on-board the robot. It would be efficient to run ROS related algorithms on the Jetson nano and the applications like Unity game engine, visualization tools related to ROS should be run on a computer. The robot also has its own router which easily allows for multiple machine communication.

The Unity game engine is integrated with IBM Watson, an Artificial intelligence that is needed for text to speech and speech to text conversion. The system in this experiment supports a limited number of questions listed in IBM Watson's dialogue system. The IBM Watson is always listening to the microphone and even a simple conversation that is not directed towards the robot can trigger the avatar system and the robot stops moving. The robot starts again after initiating a start sequence. However, this can be addressed by handling the input to the microphone at the application level. For example, allowing the avatar to trigger the conversations with humans when certain vocabulary is used like "Hello Jason". This will allow the robot to have a limited level of conversations but has the advantage of not stopping the robot randomly while navigating.

The figure 5.4 shows the entire system with the avatar, navigation on the prebuilt map, and the camera detecting an April tag. The entire system is able to maneuver the map built from the floorplan. navigating from one tag to another explaining about the labs.

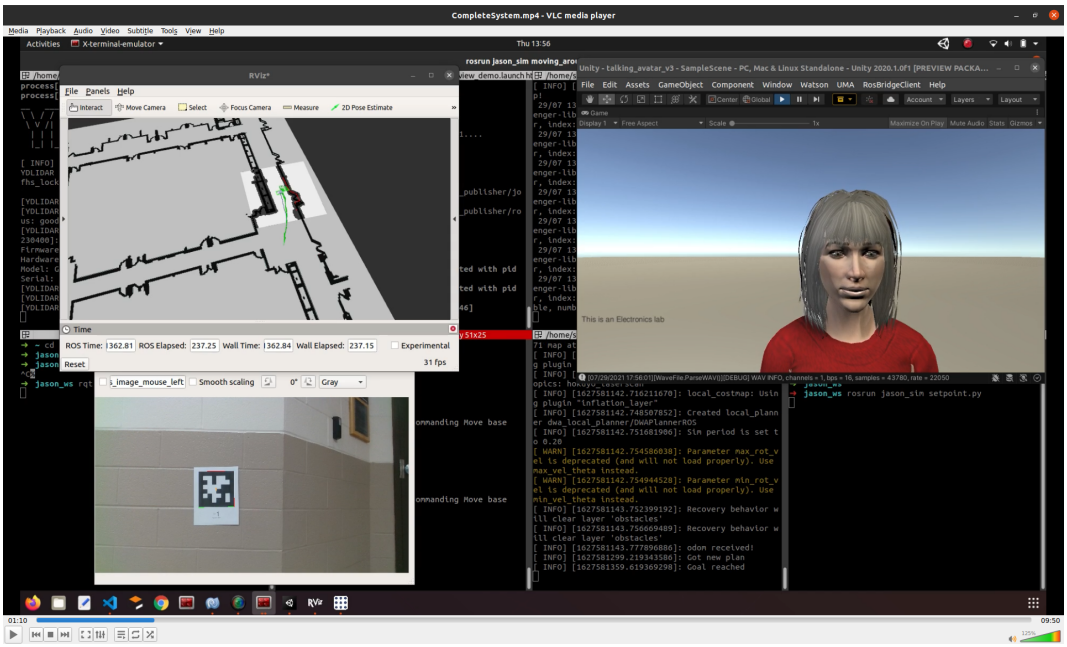

Figure 5.4: Screenshot of the entire system

Tour guide robot trials	**Time Taken in seconds (s)**
Trial 1	10686
Trial 2	10069
Trial 3	9847

Table 5.3: Tour guide trials

Each trial in the table 5.3 represents the time taken for the robot to provide tour by traversing 3 different April tags. As observed from the table, all the three trials took approximately same time for the robot to provide tour.

Chapter 6

Conclusion

This experiment deals with the system integration of an Autonomous Tour Guide robot. The self-driving nature of the robot is implemented using the ROS navigation stack. The Avatar system with lip sync is handled by Unity and IBM Watson is used for limited interaction with humans.

The robot in this experiment is able to detect obstacles and drive past them by avoiding a collision. The Environment contains April tags at known locations which are stuck to the laboratories and the robot drives from one April tag location to another describing the experiments that are going behind those doors to the people. IBM Watson, Unity, and ROS interact using existing libraries that use web sockets.

The robot's effectiveness in operating as a tour guide robot is tested on the 3rd floor of KGCOE, RIT in the department of Electrical and Micro electrical engineering which consists of many laboratories with corridors.

Chapter 7

Future Work

In the above-conducted experiment, the robot is integrated with multiple systems. They are ROS navigation stack, Unity's Avatar system, and IBM Watson for speech recognition system.

Figure 7.1: Laboratory Label

In this experiment, for the robot to provide a tour of the environment, It has to be supported with visual fiducial markers such as April tags. However, In RIT, each laboratory has its own labels that are stuck beside the door. The robot can use OpenCV techniques to detect those labels and provide information about the experiments going in those labs. The camera mounted on the robot could be adjusted so as to not turn the robot as much to detect

tags.

All subsystems on the robot are running on a laptop. ROS and Unity 3D game engine are battery-hungry applications. In the future, ROS could be run on a single-board computer like Nvidia Jetson Nano, which will handle all the applications needed to drive the robot autonomously. The laptop can run the Unity game engine and visualization tools of ROS.

Chapter 8

System Architecture on other robots

This chapter deals with the implementation and integration of the systems used in this experiment on other robots. Any robot that implements the architecture of tour guide robot in this experiment needs an odometry source, 2D laser scan and a camera.

ROS navigation stack brings the autonomous capability to any wheeled and legged robots with the proper initial setup of lower-level hardware and controls. Omni-directional, differential drive, steering robots are some of the wheeled robots that have support for autonomous functionality in ROS.

April tags have to be set up in the environment at known locations so that robots can maneuver to these tags positions.

The rest of the setup is as explained in the chapter 3 of this experiment.

Bibliography

[1] Z. Xuexi, L. Guokun, F. Genping, X. Dongliang and L. Shiliu, "SLAM Algorithm Analysis of Mobile Robot Based on Lidar," 2019 Chinese Control Conference (CCC), 2019, pp. 4739-4745, doi: 10.23919/ChiCC.2019.8866200.

[2] E. Olson, "AprilTag: A robust and flexible visual fiducial system," 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 3400-3407, doi: 10.1109/ICRA.2011.5979561.

[3] R. Codd-Downey, P. M. Forooshani, A. Speers, H. Wang and M. Jenkin, "From ROS to unity: Leveraging robot and virtual environment middleware for immersive teleoperation," 2014 IEEE International Conference on Information and Automation (ICIA), 2014, pp. 932-936, doi: 10.1109/ICInfA.2014.6932785.

[4] S. E. Hadji, T. H. Hing, M. S. M. Ali, M. A. Khattak and S. Kazi, "2D occupancy grid mapping with inverse range sensor model," 2015 10th Asian Control Conference (ASCC), 2015, pp. 1-6, doi: 10.1109/ASCC.2015.7244705

[5] I. Naotunna and T. Wongratanaphisan, "Comparison of ROS Local Planners with Differential Drive Heavy Robotic System," 2020 International Conference on Advanced Mechatronic Systems (ICAMechS), 2020, pp. 1-6, doi: 10.1109/ICAMechS49982.2020.9310123.

[6] D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance", IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23-33, March 1997.

[7] A. Hussein, F. García and C. Olaverri-Monreal, "ROS and Unity Based Framework for Intelligent Vehicles Control and Simulation," 2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES), 2018, pp. 1-6, doi: 10.1109/ICVES.2018.8519522

[8] S. Thrun et al., "MINERVA: a second-generation museum tour-guide robot," Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), 1999, pp. 1999-2005 vol.3, doi: 10.1109/ROBOT.1999.770401.

[9] Harihar Subramanyam et al. "SKYCALL", [online]. https://senseable.mit.edu/skycall/

[10] Yuri Kageyama, "Honda robot Asimo makes balky tour guide", [online]. https://www.usatoday.com/story/driveon/2013/07/06/honda-robot-asimo/2494143/

[11] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," [1993] Proceedings IEEE International Conference on Robotics and Automation, 1993, pp. 802-807 vol.2, doi: 10.1109/ROBOT.1993.291936.

[12] Automaticaddison, "How to Create a Map for ROS From a Floor Plan or Blueprint", [online]. https://automaticaddison.com/how-to-create-a-map-for-ros-from-a-floor-plan-or-blueprint/

[13] C. Kim, R. Sakthivel and W. K. Chung, "Unscented FastSLAM: A Robust and Efficient Solution to the SLAM Problem," in IEEE Transactions on Robotics, vol. 24, no. 4, pp. 808-820, Aug. 2008.

[14] Jakob, "Difference between Rao-Blackwellized particle filters and regular ones", [online]. https://robotics.stackexchange.com/questions/2251/difference-between-rao-blackwellized-particle-filters-and-regular-ones

[15] R. E. Kalman, "A new approach to linear filtering and prediction problems", Journal of Fluids Engineering, vol. 82, no. 1, pp. 35-45, 1960

[16] A. B. S. H. M. Saman and A. H. Lotfy, "An implementation of SLAM with extended Kalman filter," 2016 6th International Conference on Intelligent and Advanced Systems (ICIAS), Kuala Lumpur, 2016, pp. 1-4.

[17] S. J. Julier, J. K. Uhlmann, "New extension of the kalman filter to nonlinear systems", AeroSense97. International Society for Optics and Photonics, pp. 182-193, 1997.

[18] S. Ön and A. Yazici, "A comparative study of smooth path planning for a mobile robot considering kinematic constraints," 2011 International Symposium on Innovations in Intelligent Systems and Applications, Istanbul, 2011, pp. 565-569.

[19] O. Ozisik and S. Yavuz, "An Occupancy Grid Based SLAM Method," 2008 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications, Istanbul, 2008, pp. 117-119.

[20] S. Chan, P. Wu and L. Fu, "Robust 2D Indoor Localization Through Laser SLAM and

Visual SLAM Fusion," 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 2018, pp. 1263-1268.

[21] Babinec, Andrej & Jurišica, Ladislav & Hubinský, Peter & Duchoň, František. (2014). Visual Localization of Mobile Robot Using Artificial Markers. Procedia Engineering. 96. 10.1016/j.proeng.2014.12.091.

[22] A. Al-Wazzan, R. Al-Farhan, F. Al-Ali and M. El-Abd, "Tour-guide robot," 2016 International Conference on Industrial Informatics and Computer Systems (CIICS), Sharjah, 2016, pp. 1-5.

[23] Woojin Chung, Gunbee Kim, Munsang Kim and Chongwon Lee, "Integrated navigation system for indoor service robots in large-scale environments," IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004, New Orleans, LA, USA, 2004, pp. 5099-5104.

[24] A. B. S. H. M. Saman and A. H. Lotfy, "An implementation of SLAM with extended Kalman filter," 2016 6th International Conference on Intelligent and Advanced Systems (ICIAS), Kuala Lumpur, 2016, pp. 1-4.

[25] S. Rawat, P. Gupta and P. Kumar, "Digital life assistant using automated speech recognition," 2014 Innovative Applications of Computational Intelligence on Power, Energy and Controls with their impact on Humanity (CIPECH), Ghaziabad, 2014, pp. 43-47.

[26] C. Kim, R. Sakthivel and W. K. Chung, "Unscented FastSLAM: A Robust and Efficient Solution to the SLAM Problem," in IEEE Transactions on Robotics, vol. 24, no. 4, pp. 808-820, Aug. 2008.

[27] S. Julier, "The scaled unscented transformation", Proc. Amer. Control Conf., pp. 4555-4559, 2002.

[28] R. Merwe, A. Doucet, N. Freitas, E. Wan, The unscented particle filter.

[29] D. Lee, D. Kim, S. Lee, H. Myung and H. Choi, "Experiments on localization of an AUV using graph-based SLAM," 2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, 2013, pp. 526-527.

[30] R. Mur-Artal, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," in IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147-1163, Oct. 2015.

[31] ROS, "Robot Operating System" [online]. https://www.ros.org/about-ros/

[32] Madgwick, Sebastian & Harrison, Andrew & Vaidyanathan, Ravi. (2011). Estimation of IMU and MARG orientation using a gradient descent algorithm. IEEE ... International Conference on Rehabilitation Robotics : [proceedings]. 2011. 5975346. 10.1109/ICORR.2011.5975346.

[33] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. 2005. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press.

[34] Dellaert, F.; Fox, D.; Burgard, W.; and Thrun, S. 1999b. Monte Carlo localization for mobile robots. Proc. of ICRA-99

[35] Giannoulis, George & Kamarinos, George & Roussou, Maria & Trahanias, Panos & Argyros, Antonis & Tsakiris, Dimitris & Cremers, Armin & Schulz, Dirk & Haehnel,

Dirk & Savvaides, Vassilis & Friess, Peter & Konstantios, Dimitrios. (2001). Enhancing Museum Visitor Access Through Robotic Avatars Connected to the Web.

[36] Thrun, S., Maren Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hähnel, G. Lakemeyer, Charles J. Rosenberg, N. Roy, Jamieson Schulte, D. Schulz and W. Steiner. “Experiences with two Deployed Interactive Tour-Guide Robots.” (1999).

[37] f-jiang, "a solution to a strange issue with interrupts quadrature encoder" [Online], https://forum.arduino.cc/t/strange-issue-with-interrupts-quadrature-encoder/651178.

[38] ROS Documentation, "Navigation stack" [Online]. http://wiki.ros.org/navigation/Tutorials/RobotSetup.

[39] Eitan Marder-Eppstein & David V. Lu!! & Dave Hershberger "Costmap_2D" [Online]. http://wiki.ros.org/costmap_2d"

[40] Honda, "History of ASIMO Development" [Online]. https://global.honda/innovation/robotics/ASIMO/history.html

[41] Tony_Pigram, "Create a 3D Digital Human with IBM Watson Assistant and Unity3D" [Online]. https://developer.ibm.com/recipes/tutorials/create-a-3d-digital-human-with-ibm-watson-assistant-and-unity3d/

[42] YDLIDAR, "YDLIDAR G4" [online]. https://www.ydlidar.com/products/view/3.html

[43] Intel, "Intel® RealSense™ Depth Camera D435i" [online]. https://store.

intelrealsense.com/buy-intel-realsense-depth-camera-d435i.html?_ga=2.99806238.1689216786.1627919568-1648489121.1582330121

[44] H. AlShu'eili, G. S. Gupta and S. Mukhopadhyay, "Voice recognition based wireless home automation system," 2011 4th International Conference on Mechatronics (ICOM), 2011, pp. 1-6, doi: 10.1109/ICOM.2011.5937116.

[45] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai and R. Vincent, "Efficient Sparse Pose Adjustment for 2D mapping," 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010, pp. 22-29, doi: 10.1109/IROS.2010.5649043.